Programación mediante tarjetas gráficas
Mejora de una aplicación distribuida

Autor: Samuel Rodríguez Sevilla
Tutor: José María Sierra Cámara

25 de febrero de 2011

Jesús le preguntó: «¿Cuál es tu nombre?»

«Legión», respondió, porque eran muchos los demonios que habían entrado en él.

Lucas 8-30

Resumen

El presente proyecto final de carrera ha consistido en el diseño y la modificación de una herramienta de evaluación de seguridad especializada en la comprobación de contraseñas utilizando un sistema de fuerza bruta. Las modificaciones realizadas sobre la herramienta dotan a ésta de nuevas características. Estas han consistido en la mejora de su usabilidad, la capacidad de ampliar fácilmente sus funcionalidades y conseguir que las tareas de mantenimiento sean más sencillas. Con todo ello se pretende obtener una herramienta que pueda ser utilizada con independencia de las mejoras que pueden aparecer en el marco de la seguridad utilizando contraseñas y, como no, de cualquier otro tipo de mecanismo de protección.

Palabras clave— seguridad informática, contraseñas, computación gráfica, sistema distribuido

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Uno de los puntos más débiles en la seguridad de toda organización son las contraseñas. Cuando el usuario dispone de absoluto control sobre la decisión de cómo debe ser ésta, las contraseñas tienden a ser muy débiles ante ataques de diccionario. La solución a este problema reside en el uso de políticas de contraseña que exijan unos mínimos de fortaleza (uso de minúsculas y mayúsculas, números y símbolos). Estas políticas se pueden controlar informáticamente obligando al usuario a poner contraseñas de calidad.

Las contraseñas, por seguridad, no se almacenan tal cual en los equipos sino que son tratadas con unas funciones resumen que generan un valor de tamaño conocido; éste es el dato que se almacena.

Por otra parte la potencia de las CPUs se ha incrementando, lo que ha supuesto la necesidad de mejorar los sistemas criptográficos para hacerlos más resistentes a todo tipo de ataques. En el caso concreto de las funciones resumen, este fortalecimiento se ha materializado de dos formas distintas:

- Se ha incrementado el número de bits del resumen (para disminuir la posibilidad de colisiones).
- Se procura mejorar la técnica de generación del resumen para garantizar que los resúmenes sean lo más aleatorios posibles, utilizando procesos que impidan determinar el mensaje a partir del resumen y que garanticen estar normalmente[1] distribuidos.

[1]Si los resúmenes generados no tuviesen una distribución normal esto supondría que habría grupos de resúmenes y facilitaría ataques contra éstos pues sería más fácil determinar de dónde podrían proceder.

Estas mejoras han ido esquivando uno de los problemas más importantes que han tenido las funciones resumen: el incremento de la potencia de las CPUs permite realizar ataques de fuerza bruta en tiempos asumibles.

El incremento de la potencia de las CPUs se ha visto restringido por la capacidad de integración de transistores en un microprocesador y el diseño interno del mismo. Si tomamos la Ley de Moore, ésta dice que el número de transistores en un microprocesador tiende a duplicarse cada 18 meses. Esto nos permite planificar qué capacidad de cómputo podría tener una CPU en el futuro de cara a evitar ataques de fuerza bruta. De todos modos, el número de transistores es solo una parte de la capacidad de una CPU ya que su estructura interna afecta también de forma muy clara. Esto se debe a la forma que tiene de organizar la ejecución de una aplicación, la calidad de la predicción de saltos, etc.

Al mismo tiempo a la mejora de las CPUs, se ha ido produciendo una importante mejora en los procesadores de las tarjetas gráficas (en adelante GPU). Estos procesadores de propósito específico han visto su potencia incrementada muy rápidamente gracias a sus diseños más sencillos (se utilizan específicamente para cálculo matemático) y a su gran capacidad de paralelización (una NVIDIA Tesla puede disponer de hasta 960 núcleos) como puede verse en la figura 1.1. Desde hace algún tiempo las tarjetas gráficas permiten cargar pequeños trozos de códigos denominados *shaders* (estos códigos se utilizan como filtros para efectos 3D) y esta técnica a desembocado en permitir la carga de códigos de usuario más genéricos.

La principal razón por la que la computación utilizando GPUs se ha convertido en algo de interés ha sido la publicación de APIs como CUDA y OpenCl que nos permiten aprovechar las capacidades de las actuales tarjetas gráficas para realizar cálculos y operaciones muy costosas en tiempo de CPU de forma más rápida. Además, permiten un alto grado de paralelismo, lo que puede resultar beneficioso para el desarrollo de ciertos tipos de programas.

En la actualidad, mientras con una CPU se podía conseguir hasta 80 millones de claves MD5 por segundo (aplicando muchas optimizaciones a nivel de lenguaje ensamblador), una GPU potente puede alcanzar hasta cerca de los 2.000 millones de resúmenes por segundo. Esto es, una GPU es hasta 250 veces más rápida que la CPU para este tipo de cálculos, propiciado, principalmente, por su mayor capacidad de paralelización.

Si consideramos el hecho de que en la actualidad casi todos los nuevos equipos que se venden en el mercado disponen de aceleradoras gráficas y que es muy fácil crear una red de ordenadores zombis se debe considerar

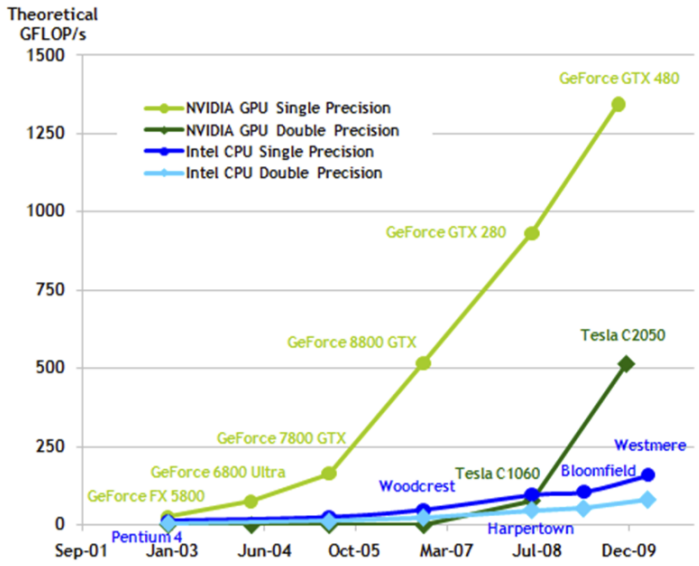

Figura 1.1: *Comparativa de la evolución de los GFlops de las GPUs frente a las CPUs[NVI10]*

la mejora de los mecanismos de contraseña una prioridad por parte de las organizaciones. Por este motivo es importante disponer de herramientas que permitan comprobar la fortaleza de las contraseñas y evaluar la dificultad de realizar ataques a los distintos algoritmos de resumen existentes y en experimentación.

1.1. Motivación

Como ya se ha comentado, es importante garantizar la calidad de los algoritmos que se utilizan en seguridad y las contraseñas empleadas dentro de las organizaciones. Por este motivo nace este proyecto fin de carrera que pretende ser el inicio de una herramienta versátil, sencilla y atractiva que permita ofrecer a las distintas organizaciones un mecanismo para fortalecer su seguridad. Por otra parte, el estudio del uso de tarjetas gráficas fuera del ámbito de los videojuegos y del diseño gráfico es de gran interés por la alta capacidad de cómputo que ofrecen éstas. De este modo se pretende garantizar que la amortización de los equipos informáticos es máxima ya que el aprovechamiento de los mismo será mayor. Hay que recordar que en la actualidad todos los ordenadores disponen de tarjetas gráficas con aceleración

3D y de las 3 grandes marcas del mercado (Intel, AMD y NVIDIA), dos de ellas ofrecen desde hace unos años la capacidad de ejecutar código de usuario en las mismas. En el caso de Intel, no se ofrece el uso de su GPU para cálculo, pero si permiten programar utilizando OpenCl (ver el capítulo 3) para sus procesadores.

Por otra parte, la popularización de *frameworks* MVC (se basan en el patrón de arquitectónico MVC) han revolucionado el mundo de la programación web. Ejemplos de ello los tenemos en Ruby on Rails, DJango o CakePHP. Lo interesante de estos *frameworks* es que mejoran sustancialmente las tareas de mantenimiento de los programas web ofreciendo una serie de pautas tanto en la nomenclatura de variables y tablas de las bases de datos como en la organización que deben tener los ficheros de código fuente.

1.2. Objetivos

Para poder comprobar la fortaleza de una contraseña dada y para poder evaluar la resistencia frente a ataques de fuerza bruta de diferentes tipos de funciones resumen hace falta disponer de una herramienta adecuada. Para este fin existe una gran cantidad de herramientas, principalmente destinadas a la recuperación de contraseñas extraviadas, pero que su código es cerrado y su precio elevado. Aunque el precio puede no resultar un problema importante, consideramos que el uso de aplicaciones que no suministran su código fuente no es recomendable en el ámbito de la seguridad informática ya que es importante poder saber qué es lo que la herramienta hace y eso solo se puede determinar de forma fehaciente si ésta es de código abierto. Por otra parte, también es recomendable que la herramienta sea software libre porque así, en caso de que el fabricante deje de dar soporte a la misma, tendríamos permiso para poder realizar modificaciones eliminando de este modo la dependencia con el vendedor.

El presente proyecto final de carrera nace con el objetivo de aprovechar las nuevas arquitecturas gráficas en el entorno de la evaluación de seguridad informática, especialmente en el ámbito de las funciones resumen. Igualmente se ha pretendido utilizar un modelo distribuido para mejorar la escalabilidad del sistema y así disponer de una herramienta potente que pueda ser ampliada según la necesidad del momento.

En concreto, los objetivos que se pretenden alcanzar con la realización de este proyecto son:

Estudio de las distintas técnicas de evaluación de contraseñas. Para poder garantizar la calidad de cualquier solución planteada para la evaluación de contraseñas es necesario disponer de una base suficiente sobre el tema. De este modo será más sencillo determinar si el trabajo realizado es correcto o no.

Aprendizaje de programación con GPU. Esto permitirá hacer uso de un elemento de gran potencia que se encuentra hoy día en todos los ordenadores comercializados. Además, las GPU ofrecen unas características ideales para la ejecución de una gran cantidad de tareas en paralelo lo que las hace muy atractivas para realizar tareas que de otro modo podrían tardar muchísimo.

Utilizar nuevas técnicas de distribución de tareas. A lo largo de la carrera se ha tratado en muchas ocasiones los sistemas distribuidos y cómo estos distribuyen su carga de trabajo entre conjunto de ordenadores. En este punto se pretende experimentar con algunas técnicas no tratadas o con mecanismos diferentes a los vistos.

Aplicar técnicas de usabilidad. Un apartado importante de toda aplicación es la facilidad con la que esta se maneje y el grado de satisfacción del usuario de la misma. Por este motivo es importante dedicar un esfuerzo en el diseño de la interfaz y así mejorar la experiencia del usuario.

Ofrecer una buen grado de escalabilidad. Gracias a esto se pretende conseguir que el sistema desarrollado pueda crecer con el tiempo de forma sencilla introduciendo nuevos ordenadores. De este modo se reducirían los costes al no tener que comprar grandes sistemas exclusivamente para este propósito.

Garantizar que la herramienta desarrollada sea mantenible. Para ello es necesario hacer un buen uso de los comentarios y se debe garantizar que estén bien documentados los aspectos más importantes de diseño de la herramienta. De este modo se conseguirá que en un futuro cualquier persona pueda realizar modificaciones sobre la misma en caso de fallos.

Implementar un sistema de extensiones. Con esto se pretende que la herramienta pueda ver ampliada su funcionalidad sin necesidad de que

deba ser compilada nuevamente. Para ser más exactos, lo que se pretende es que, durante el inicio de la ejecución de la herramienta, ésta busque en el sistema extensiones que le añadan funcionalidades.

Crear un mecanismo sencillo de administración. De este modo se pretende mejorar el tiempo requerido para la realización de las tareas relacionadas con la aplicación.

Desarrollo de una interfaz web. En la actualidad casi todos los equipos informáticos se encuentran conectados, sino a Internet, sí a una red de ordenadores. Por norma general los cortafuegos empresariales filtran el tráfico que circula por la red, pero permiten el uso de Web lo que convierte a la Web casi en el medio exclusivo para usarlo como protocolo para la administración remota.

Uso de los *frameworks* MVC. Con esto se pretende garantizar la mantenibilidad de la solución web anteriormente comentada. El uso de *frameworks* ya existentes facilita el mantenimiento de la aplicación, ya que sólo habrá que dedicar esfuerzo al desarrollo de la herramienta, y en caso de mejoras en el *framework* éstas se pueden ver inmediatamente reflejadas en la aplicación sin apenas tener que realizar cambios (en muchos casos no es necesario realizar cambio alguno).

1.3. Estructura del documento

Para facilitar la búsqueda a través del este documento, se provee de la siguiente guía cona organización interna del mismo. Éste se divide en los siguientes capítulos:

Funciones resumen: introducción a las funciones resumen y a los tipos de ataques que existen sobre ellas. Es importante comprender qué son para entender mejor cuales son los objetivos del proyecto.

Desarrollo con tarjetas gráficas: se introduce la historia del desarrollo con tarjetas gráficas y a cómo se programa utilizando CUDA, el API de NVIDIA. Con esto se pretende ofrecer una visión global sobre el uso de tarjetas gráficas y el porqué de su uso.

Mejora de una aplicación distribuida: se muestra un estudio sobre una aplicación ya existente y los cambios que se le han realizado y las motivaciones para ello.

Resultados: se enumeran las pruebas y los resultados obtenidos por las mismas.

Conclusiones: conclusiones obtenidas de la realización del proyecto en las que se intenta valorar el grado de consecución del mismo.

Trabajos futuros: descripción de posibles trabajos futuros que sería interesante realizar a partir del trabajo realizado en este proyecto final de carrera.

Capítulo 2

Funciones resumen

Los sistemas destinados a ocultar o proteger información llevan usándose desde tiempos de la antigua Roma [LP87] e incluso antes. Paralelamente se ha tratado siempre de crear las técnicas necesarias para poder acceder a dicha información sin ser el destinatario legítimo de la misma. Esto ha creado la necesidad de mejorar constantemente los mecanismos de cifrado para evitar que la información protegida no pueda ser utilizada salvo por aquellos a la que está destinada.

En la actualidad hay una gran cantidad de sistemas para la protección de información y se utilizarán unos u otros dependiendo de lo que se pretenda hacer. Concretamente existen 3 grandes grupos de mecanismos de seguridad:

- Sistemas de cifrado de clave simétrica, que son aquellos que utilizan una única clave para cifrar la información y descifrar la misma. Esta clave debe ser conocida por todos aquellos que quieran tener acceso a los datos.

- Sistemas de cifrado de clave asimétrica, que utiliza dos juegos de claves, una pública y conocida por todo el mundo y otra privada que solo su propietario posee. Estos sistemas permiten cifrar y descifrar mensajes y firmar los mismos (depende del tipo de claves).

- Sistemas de un solo sentido o funciones resúmenes (también conocidas como funciones *hash*).

Las funciones resumen, que son las que nos interesan para el presento proyecto fin de carrera, son aquellas que cumplen las siguientes características:

- Son fáciles de calcular en un sentido, pero es muy complicado hallar su inversa (en principio ésta no existe).
- Dada una entrada de longitud arbitraria siempre producirán una salida de longitud fija.
- Debe ser muy complicado encontrar colisiones, esto es, dos entradas diferentes que produzcan el mismo resumen.

Este tipo de funciones son ampliamente utilizadas en el mundo de la seguridad como sistema para el almacenamiento de contraseñas de usuario, la generación de claves de sesión o la firma digital de documentos (por poner algunos ejemplos). Al ser ampliamente utilizadas es importante disponer de mecanismos para comprobar la fortaleza del mecanismo de funcionamiento. Por otra parte también hay que poder comprobar la calidad de las contraseñas para evitar problemas por posibles debilidades de las funciones resumen.

A causa de su gran uso es necesario disponer de sistemas que comprueben la fortaleza de las contraseñas elegidas por los usuarios o de las claves de sesión que pueda generar un sistema de seguridad. El primer caso es importante para garantizar la seguridad de las organizaciones, impidiendo que los usuarios elijan contraseñas que puedan ser adivinadas o quebrantadas por posibles atacantes dando acceso a la información privada de ésta con los consiguientes problemas por posibles copias y/o borrados de información. El segundo caso es importante para garantizar que los sistemas seguros sean capaces de generar claves suficientemente robustas para impedir ataques externos.

2.1. Comprobación de funciones resumen y contraseñas

Existen dos formas básicas para comprobar la fortaleza de los mecanismos de seguridad. El primero es buscar debilidades en la propia función resumen que se va a utilizar. El segundo mecanismo es comprobar la calidad de la clave utilizada. Para este último caso lo más sencillo es utilizar un mecanismo de fuerza bruta. Éste tipo de comprobación consiste en probar todas las posibles entradas para generar resúmenes y éstos se cotejan con un resumen conocido previamente.

El mayor problema de los sistemas de fuerza bruta es el tiempo que tardan en ofrecer algún resultado. Esto se debe a la gran cantidad de comprobaciones

que deben realizar. Por este motivo es importante poder predecir el tiempo que dedicarán previamente para comprobar si vale o no la pena intentar realizar una comprobación de éste tipo.

Para poder calcular el tiempo que se necesitará para hacer una comprobación de fuerza bruta empezaremos por el caso general. En éste, el tiempo máximo (el que recorre todas la combinaciones posibles) es de:

$$T_{max} = t \sum_{i=m}^{n} k^i$$

Donde m es la longitud mínima de la entrada, n es la longitud máxima y k es el número de símbolos posibles del alfabeto a utilizar. Finalmente t representa el tiempo de cómputo de la función resumen.

Utilizando este sistema como referencia de peor caso es fácil medir las mejoras aportadas por otros algoritmos. De este modo, y tomando como referencia el trabajo realizado en este proyecto final de carrera, el simple uso de la paralelización utilizando un sistema de C procesadores homogéneos nos proporciona unos tiempos de:

$$T_{max} = t \sum_{i=m}^{n} \frac{k^i}{C}$$

En la actualidad, gracias a los avances en las comunicaciones y a los distintos procesadores es normal disponer de sistemas distribuidos heterogéneos. Estos pueden contar con unas cuantas máquinas o cientos de miles. Para este tipo de casos es necesario disponer de una función general que tenga en cuenta la velocidad de cálculo en cada tipo de procesador en que vaya a ejecutarse la función resumen. Si disponemos de p tipos de procesadores distintos y C_j procesadores para cada tipo que tardan t_j segundos en computar la función resumen ($1 \leq j \leq p$):

$$T_{max} = \frac{\sum_{i=m}^{n} k^i}{\sum_{j=1}^{p} \frac{C_j}{t_j}}$$

Con esta información se puede calcular cual debe ser el tamaño del sistema a utilizar para poder comprobar la fortaleza de una contraseña en un tiempo determinado.

Por ejemplo, si dispusiéramos de un sistema con las siguientes características:

- 4 microprocesadores capaces de hacer 25 millones de resúmenes por segundo cada uno.

- 4 tarjetas gráficas capaces de hacer 1.250 millones de resúmenes por segundo cada una.

Tardaríamos unas 9,5 horas en encontrar una contraseña de entre 6 y 8 caracteres utilizando solo letras minúsculas y números. Es importante tener en cuenta que al buscar una contraseña se puede acotar mucho el conjunto de símbolos de entrada y las longitudes de las claves. Esto reduce enormemente los tiempos de búsqueda.

Es fácil comprobar que utilizando sistemas de fuerza bruta para comprobar resúmenes se resuelve de forma lineal con respecto al número de procesadores.

A parte de los sistemas de fuerza bruta, existen muchas técnicas que han surgido a partir de la investigación de los distintos tipos de sistemas de resumen. Estos nuevos sistemas proceden de debilidades de los propios algoritmos y deben ser tenidos en cuenta a la hora de evaluar la fortaleza de las funciones resumen.

Aunque no es el propósito del presente proyecto de fin de carrera implementar todos los sistemas de comprobación conocidos sí es importante tenerlos en cuenta para poder hacer comparaciones y valoraciones con respecto a las soluciones implementadas.

2.1.1. Ataque de cumpleaños

Este ataque a los sistemas de resúmenes consiste en que dado un mensaje M cualquiera y una función resumen H, que genera resúmenes de longitud L bits, se puede hallar un mensaje M' probando combinaciones aleatorias en aproximadamente $1{,}2\sqrt{2^L}$ intentos [Sec, vOW94](en caso de que la función resumen sea uniformemente distribuida).

El principio en el que se base este ataque se encuentra en un problema de teoría de la probabilidad conocido como problema del cumpleaños o paradoja del cumpleaños. Esta dice que un grupo de personas debe tener al menos 23 individuos para que haya al menos un 50 % de probabilidad de que dos hayan nacido el mismo día (figura 2.1).

El funcionamiento general es el siguiente:

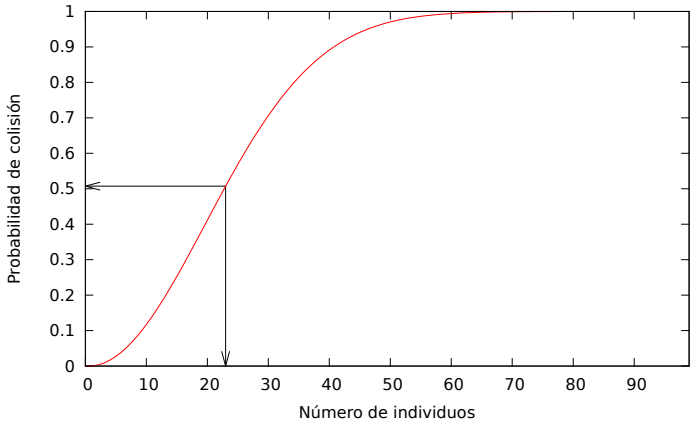

Figura 2.1: *Probabilidad de encontrar dos personas nacidas el mismo día con respecto al tamaño del grupo*

$$P(i,n) = \begin{cases} 1 & \text{si } i = 1 \\ P(i-1,n)\frac{n-i+1}{n} & \text{si } i > 1 \end{cases}$$

Donde $P(i,n)$ es la probabilidad de que para una población final de n elementos (365 en el caso de los cumpleaños), y disponiendo de i elementos seleccionados aleatoriamente, dos individuos no coincidan. Por ejemplo, si tomamos tres individuos aleatoriamente el primero de ellos habrá nacido un día cualquiera del año (la probabilidad de que dos individuos no cumplan años el mismo día es del 100 % ó 365/365), el segundo habrá nacido uno de los restantes días (364/365) y el tercero igual (353/365). La probabilidad total será el producto de las probabilidades; en este caso concreto es de 0,9945.

Este mismo principio es aplicable a las funciones resumen [BK04], ya que se puede considerar que en lugar de días del año tenemos resúmenes (todas las posibles combinaciones de éstos) y en lugar de personas tenemos textos o mensajes a ser cifrados.

En el caso de las funciones resumen hay que tener en cuenta la distribución que hacen éstas de los datos de entrada ya que las funciones no uniformes serán en las que es más sencillo encontrar colisiones (se puede centrar la búsqueda en los cúmulos). Esto supone que en lugar de tener que probar combinaciones aleatorias diferentes podríamos reducir el número de intentos.

Estas características del sistema del cumpleaños lo convierten en un sistema ideal para sustituir a los mecanismos de fuerza bruta convencionales, pero hay que tener en cuenta que se debe considerar el tiempo de crear los

M	$H(M)$	$H(M')$	$H(M'')$	$H(M''')$
m_1	m'_1	m''_1	m'''_1	$m_{1,terminal}$
m_2	m'_2	m''_2	m'''_2	$m_{2,terminal}$
m_3	m'_3	m''_3	m'''_3	$m_{3,terminal}$
m_4	m'_4	m''_4	m'''_4	$m_{4,terminal}$

Cuadro 2.1: *Ejemplo de generación de una tabla arcoíris*

mensajes a probar (dependiendo de su longitud podría hacer al sistema igual de rápido que uno de fuerza bruta) y el tamaño de la muestra. El principal problema del ataque de cumpleaños es que debe trabajar sobre todas las posibles combinaciones existentes, lo que puede suponer un problema en casos en los que sepamos que hay una gran cantidad de restricciones. Por ejemplo, en el caso expuesto al inicio de la sección en el que se tardaba 9,5 horas en completar la búsqueda se tardaría cerca de 8.217,73 años utilizando este mecanismo. Esto significa que hay que estudiar el problema para poder descubrir las debilidades locales (en el caso de las contraseñas puede ser la longitud, el conjunto de caracteres posibles, etc.).

2.1.2. Tablas arcoíris

Las tablas arcoíris son una técnica por la que se toma un conjunto de entradas y sobre cada una de éstas se realizan un proceso iterativo de resúmenes que van tomando el último resumen realizado para hacer uno nuevo hasta obtener un elemento que consideraremos terminal. Se puede considerar este proceso como una tabla donde la primera columna representa las distintas entradas que se han elegido y cada columna C_j, para $j > 1$, representa el resumen de la columna anterior ($C_j = H(C_{j-1})$, ver cuadro 2.1). La tabla arcoíris consiste en tomar la primera y última columnas.

Una vez que se dispone de la tabla arcoíris se puede empezar a probar claves por fuerza bruta. Si en algún momento alguna coincide con algún elemento final de la tabla arcoíris simplemente tenemos que buscar los resúmenes empezando en el que genero dicho elemento terminal (el mensaje inicial).

El principal problema de éste sistema es determinar el tamaño de la tabla a utilizar y el tiempo que se necesita para crearla. En general este mecanismo solo se utiliza para casos muy concretos de funciones resumen débiles.

2.1.3. Caminos diferenciales para SHA-1

La técnica de los caminos diferenciales consiste en realizar un estudio sobre los cambios en los bits de las variables utilizadas en una función resumen. El objetivo es detectar estadísticamente los puntos en los que se producen alternancia de bits o en los que éstos no cambian para realizar una aproximación estadística a posibles colisiones.

En [MHP00] se puede encontrar un estudio que a partir del uso de aproximaciones no lineales es capaz de encontrar una colisión en SHA-1 en 2^{52} intentos. Para ello toma el resumen para el que deseamos hallar un mensaje que lo genere y trata de determinar los mensajes que pudieron haberlo generado deshaciendo el algoritmo tomando los datos de los posibles cambios obtenidos en el estudio anteriormente comentado.

Como sucedía con el ataque del cumpleaños, esta técnica es interesante siempre y cuando el conjunto de caracteres de entrada sea completo ya que en el caso de contraseñas, donde se utiliza un juego de caracteres muy limitado, una técnica de fuerza bruta puede resultar más eficiente.

Capítulo 3

Desarrollo con tarjetas gráficas

3.1. Breve historia de la computación con GPUs

Desde hace bastante tiempo se lleva usando las capacidades de las tarjetas gráficas para realizar computación. Concretamente se han utilizado para la realización de efectos sobre texturas y polígonos con la tecnología denominada *shaders*. Estos *shaders* son pequeños programas que transforman la forma de verse los puntos o como se transforman los polígonos y que se han estado ejecutando en las GPUs para mejorar su rendimiento.

A partir de esta tecnología y tras numerosas especulaciones sobre las capacidades de las tarjetas gráficas para realizar cálculos más genéricos, las compañías empezaron a abrir sus sistemas para permitir cargar códigos orientados a cualquier tipo de cálculo matemático.

Los primeros productos con este tipo de tecnología provinieron de BionicFX y fueron presentados a principios de 2005 [Tec04]. En este caso se hacía uso de gráficas con GPU NVIDIA para procesador de audio. Para ello transformaba el sonido en datos gráficos y luego éstos se procesaban con los medios matemáticos de que disponía la GPU.

Más tarde, en el año 2006, la compañía AMD anuncia la comercialización de AMD Stream Processor [AMD06], el primer sistema de cálculo basado en tarjetas gráficas. De este modo se empezó a comercializar el primer sistema que realmente permitía cargar código de usuario en la tarjeta para realizar cálculos que no estuviesen directamente relacionados con generación de imagen en 3D.

La compañía NVIDIA publica el 15 de febrero de 2007 la biblioteca CU-

DA [NVI07] que permite usar sus tarjetas gráficas para cálculo y, además, comercializa la arquitectura Tesla que, como en el caso de ATI, es un sistema específico de cálculo basado en tarjetas gráficas.

En base a esta popularización del cómputo con tarjetas gráficas y por su utilizad gran utilidad para todo tipo de procesos (especialmente aquellos destinados a multimedia e investigación), Apple publica un borrador de OpenCl (Open Computing Language) y posteriormente éste es pasado a control de Khronos Group [Gro08]. OpenCl es el primer estándar creado específicamente para cómputo distribuido, orientado principalmente a GPUs, y que ofrece una interfaz común para todo tipo de tarjetas gráficas y otros sistemas de cálculo (como procesadores multinúcleo, FPGAs, procesadores CELL, etc.). Esta tecnología solo se incluye de serie en el sistema operativo Apple Mac OS X 10.6, pero fabricantes como ATI y NVIDIA proveen de controladores para otras plataformas.

El mecanismo más eficiente para aprovechar las capacidades de una tarjeta gráfica es utilizar la API del fabricante ya que éste está optimizado para aprovechar mejor la arquitectura. Esto supone que en casos en los que la eficiencia es algo absolutamente crítico sea mejor opción frente al uso de la biblioteca OpenCl.

Estos sistemas están teniendo mucha relevancia debido a su alto rendimiento, especialmente en aplicaciones científicas. Además, se han realizado avances en el desarrollo de aplicaciones específicas de ruptura de contraseñas.

En la actualidad la computación con tarjetas gráficas está empezando a utilizarse en todo tipo de cálculos, tanto para investigaciones científicas como para herramientas de usuario como descompresores de vídeo y audio, filtros gráficos en herramientas de diseño o videojuegos. Todo esto gracias a las grandes capacidades de paralelización de cálculos de las GPUs como a la facilidad de realizar cálculos vectoriales de forma sencilla. Esto significa que es una tecnología apoyada por la industria y que se va a disponer de soporte y documentación para realizar desarrollos con la misma.

Se puede comprobar, además, como en el año 2008 empezaron a surgir los primeros sistemas orientados a la seguridad informática que se apoyaban en el uso de tarjetas gráficas para realizar dicha función. Un ejemplo de esto lo tenemos en la herramienta de Elcomsoft publicada en octubre de 2008 [Ltd08] que hace uso de tarjetas gráficas para recuperar contraseñas.

3.2. Introducción a CUDA

Para desarrollar este proyecto se ha hecho uso de la tecnología CUDA de NVIDIA por varios motivos:

- Las tarjetas gráficas NVIDIA están muy distribuidas y vienen de serie en la mayor parte de equipos informáticos de gama media/alta.
- El uso de un API específico ayuda a aprovechar mejor las características de la arquitectura frente a un API más general como pueda ser OpenCl.
- En el momento de realizar este proyecto se dispone de un sistema NVIDIA Tesla, por lo que CUDA se convierte en la solución ideal.

A la hora de desarrollar en una nueva arquitectura es importante conocer las características de la misma. Estas características pueden ir desde cómo se gestiona la memoria hasta qué instrucciones posee.

3.2.1. Arquitectura

Las tarjetas gráficas NVIDA disponen de una gran cantidad de unidades aritmetico-lógicas (figura 3.1) para poder realizar una mayor cantidad de cálculos por ciclo de reloj [NVI10]. Gracias a esto las tareas que hacen usos de cálculos intensivos pueden verse muy beneficiadas. Por otra parte, las GPU disponen también de un gran número de unidades de control lo que permite disponer de un gran número de tareas en paralelo. La combinación de las dos características anteriores es lo que dota a las tarjetas gráficas de una gran capacidad para ejecutar algoritmos en paralelo.

Además de como está organizada la GPU, es importante tener en cuenta como se organiza la memoria (figura 3.2) ya que el buen uso de ésta influirá de forma muy significativa en el rendimiento de los programas. Esta arquitectura es la misma que la de las tarjetas gráficas y se puede representar como una pirámide de tiempos dependiendo del tipo de memoria a la que se vaya a acceder.

La memoria de sistema es la que se encuentra en el equipo sin contar la que aporta la tarjeta gráfica. Los programas de ordenador puede hacer uso de ésta de forma sencilla, pero los programas que se ejecutan dentro de la GPU no puede acceder a ella. Tanto la memoria global como la memoria de textura se encuentran en la tarjeta gráfica y servirán de almacén de los datos que

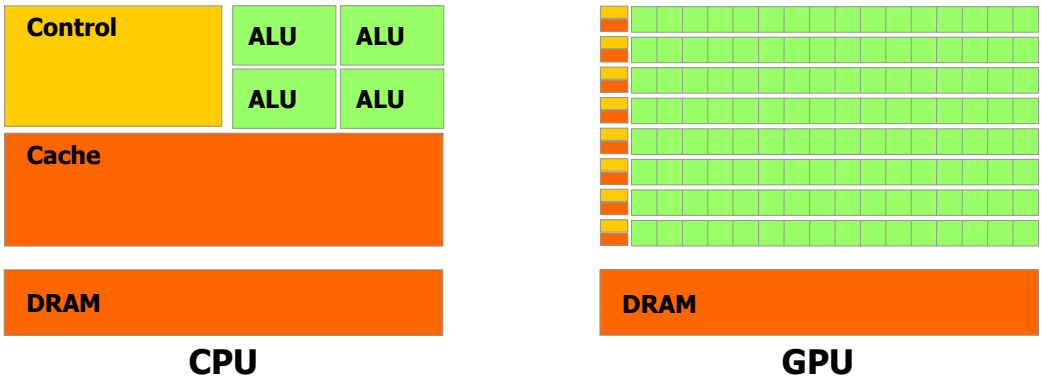

Figura 3.1: *Comparativa de organización de CPU y GPU [NVI10]*

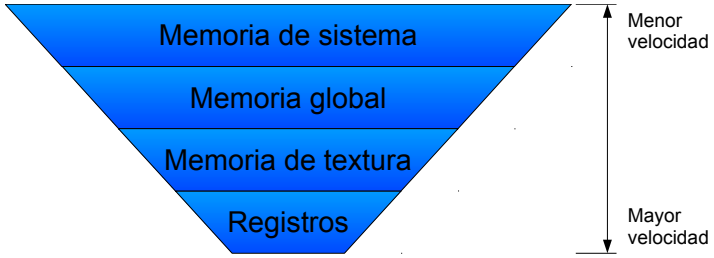

Figura 3.2: *Jerarquía de memoria en CUDA*

requieran los programas que se hallen en ésta. La principal diferencia entre ambas reside en el modo a través del cual se accede a ellas. La memoria global funciona igual que la memoria de sistema para un programa normal (a partir de un puntero se accede a la misma), mientras que la memoria de textura se accede haciendo uso de un sistema especial que añade funcionalidades como la interpolación de los valores. La ventaja de la memoria de textura es que dispone de un acceso más rápido (se debe a que es guardada en caché) lo que la convierte en una opción muy atractiva para cierto tipo de cálculos.

Por último, los registros de la GPU disponen de un acceso muy rápido por lo que es recomendable hacer uso de los mismos siempre que sea posible.

La optimización del uso de la memoria es fundamental para mejorar el rendimiento de las aplicaciones CUDA ya que si se trabaja con una cantidad muy grande de datos se puede perder una parte importante de tiempo realizando copias de memoria. Por este motivo hay que realizar un buen estudio sobre el uso que se va a realizar de la memoria.

3.2.2. Modo de desarrollo

Para programar con CUDA es muy importante tener en cuenta cómo se organiza la ejecución del código ya que influirá en la forma en la que habrá que diseñar nuestros algoritmos. De éste modo, los códigos que se ejecutan en la tarjeta gráfica se organizarán del siguiente modo:

threads que son tareas que se ejecutan en paralelo y que comparten código. Los *threads* se organizan de forma tridimensional de tal modo que podría haber momentos en los que un *thread* estuviese encima, debajo o al lado de otro.

blocks que son grupos de *threads* y, al igual que éstos, también se organizan de forma tridimensional. Todos los *threads* contenidos en un *block* se ejecutarán al mismo tiempo, de modo que si la GPU no tiene capacidad para el total de *threads* solicitados por *block* devolverá un error.

grid que es un conjunto de *blocks*. Los *grids* se organizan en planos, esto es, solo dispone de 2 dimensiones.

En la figura 3.3 puede verse de forma más clara la organización de los *threads*. Es importante tener en cuenta que, como se ha dicho, todos los *threads* de un *block* se ejecutan a la vez, pero los *blocks* no tienen porque hacerlo. Este detalle es importante a la hora de ajustar el acceso a la memoria ya que cuando un *thread* accede a ésta podrá verse optimizado si lo hace de forma ordenada con el resto de *threads* del *block*.

Otro aspecto muy importante a tener en cuenta es la organización del acceso a memoria. Como ya se vio en el apartado anterior, ésta está organizada de forma jerárquica según la velocidad de acceso. Además, también es importante tener en cuenta como los *threads* acceden a la memoria ya que si lo hacen de manera organizada se puede conseguir incrementos de rendimiento. Esto último se debe a que, si por ejemplo todos los *threads* acceden a posiciones contiguas de memoria (el primer *thread* lee el primer elemento de la memoria, el segundo *thread* lee el segundo elemento, etc.) se consigue que todas las lecturas de todos los *threads* se realicen en paralelo. Por el contrario, si las lecturas se hacen de forma desordenada, éstas se realizarán de forma secuencia, malgastando de este modo un tiempo precioso.

Como la memoria del sistema es inaccesible para los *threads* es importante tener en cuenta que sólo se podrá utilizar la memoria de la tarjeta gráfica.

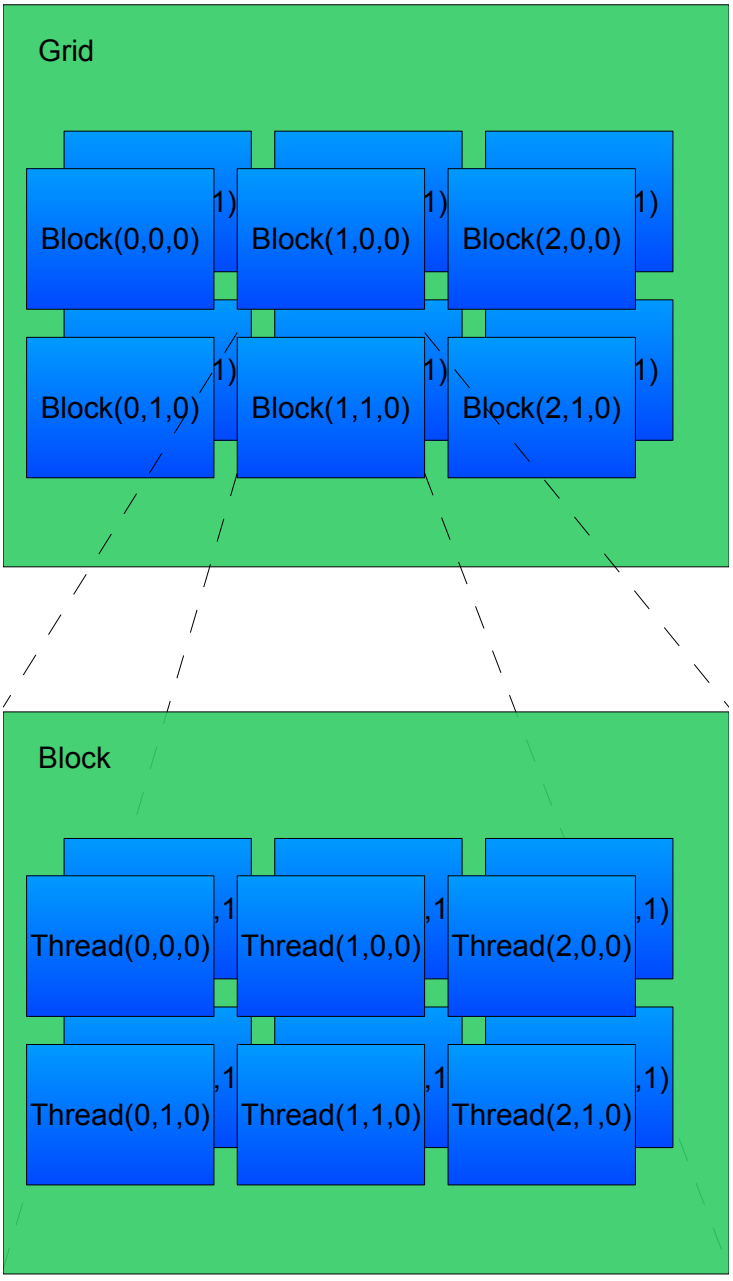

Figura 3.3: *Organización de procesos en ejecución en CUDA*

Esto supone que antes de realizar operaciones sobre memoria en CUDA hay que reservar la memoria a utilizar e inicializarla desde el programa principal que se encuentra en la CPU. Esta inicialización suele realizarse en tres tiempos:

1. En un primer momento se preparan los datos que se pasarán a la GPU.
2. Seguidamente se reserva la memoria en la tarjeta gráfica
3. Por último se copian los datos a la tarjeta gráfica para poder ser usados desde la parte que será ejecutada en la GPU.

Por otra parte, hay que tener en cuenta que una vez se disponen de los datos sobre la memoria de la tarjeta gráfica es importante estudiar si conviene utilizarlos desde dicho punto o si es preferible realizar una copia a los registros de la GPU que serán mucho más rápidos. La tarjeta gráfica provee de una gran cantidad de registros (hasta 16.384 en el caso de las NVIDIA Tesla) para poder acelerar los cálculos, por lo que se deberá hacer uso de los mismos, en la medida de lo posible. De este modo, si la cantidad de operaciones a realizar va a ser muy elevada, compensa el tiempo que se dedicará a copiar los datos desde la memoria de la tarjeta gráfica a los registros.

Por norma general, todo código que vaya a ser alojado en una GPU seguirá un patrón de ejecución como el descrito a continuación (figura 3.4):

- Se copian los parámetros que se hallen en memoria global a registros, siempre que sea posible, para acceder a los mismos desde ahí. Esto es especialmente importante si el número de accesos va a ser elevado ya que de otra forma se estaría desperdiciando una gran cantidad de tiempo en realizar accesos a memoria.
- Una vez que ya se tiene la memoria iniciada se procede a realizar los cálculos oportunos.
- Finalmente se preparan los resultados para ser volcados a la memoria global de la tarjeta gráfica y que de este modo puedan ser leídos por la aplicación.

El *kit* de desarrollo de CUDA ofrece dos formas diferentes de desarrollar aplicaciones. En la primera forma, la más sencilla, CUDA se encarga de realizar las llamadas al código que se alojará en la GPU de forma transparente

Figura 3.4: *Proceso de ejecución de un algoritmo en CUDA*

de tal forma que no tendremos que preocuparnos de configurar muchos de los parámetros de los que dispone el sistema. Este mecanismo es muy útil y permite un desarrollo rápido de funciones. Por otra parte estaría el sistema completo con el que debe utilizarse la API de bajo nivel de CUDA y que permite un nivel más alto de granularidad. Con este sistema nosotros deberemos de realizar "a mano" la carga del código en la GPU, seleccionar la GPU de todas las posibles, etc.

Con independencia del mecanismo elegido para utilizar CUDA hay algunas tareas que se deben realizar siempre de forma manual. La más importante es la administración de la memoria; cuando se va a enviar datos a la tarjeta gráfica antes de nada hay que reservar la memoria y luego se debe hacer una copia de los datos desde la memoria de sistema a la memoria que se acaba de reservar. Cuando este área de memoria ya no se necesite se deberá liberar.

El proceso de copiar memoria desde la RAM a la tarjeta gráfica es bastante rápido gracias a los nuevos buses de comunicaciones PCI Express que están especialmente diseñados para estas labores, pero esto no evita el hecho de que si la cantidad de datos a copiar es muy grande el tiempo desperdiciado entre llamadas puede ser muy grande. Esto se hace realmente patente cuando el proceso a ejecutar es muy rápido, donde se puede perder mucho tiempo realizando copias de memoria.

Por el motivo anterior CUDA provee de técnicas más avanzadas para optimizar el uso de la arquitectura y mejorar la concurrencia y así evitar tiempos muertos (figura 3.5). Éstas consisten en el uso de *streams*, por un lado, y, si es posible, la reutilización de resultados previos. Los *streams* son un mecanismo de comunicación con la tarjeta gráfica que permite tener varios canales de comunicaciones asociados con una llamada a función. Cuando se utiliza *streams* se puede disponer de dos canales, mientras se está ejecutando una función por el primer canal ya se puede utilizar el segundo canal para cargar los datos de la siguiente ejecución. Esto permite optimizar el uso de los tiempo muertos de la CPU y se ahorra la espera de la carga desde memoria de sistema a la memoria de la tarjeta gráfica.

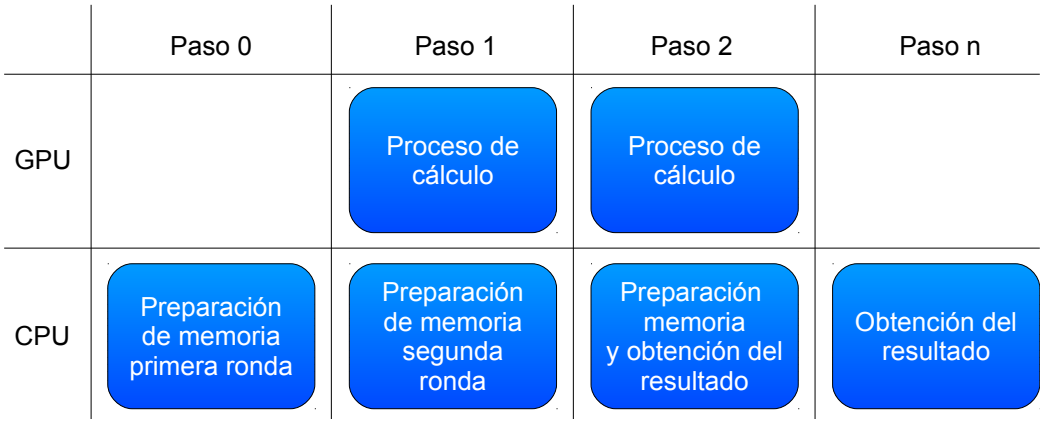

Figura 3.5: *Ejecución en GPU optimizando usando streams*

La reutilización de los valores previos es muy útil cuando se utilizan funciones de la forma $f(x) = af(x-1) + b$ ya que evita tener que realizar un exceso de copias de memoria. Si ya disponemos del resultado de la próxima ejecución en memoria simplemente lo dejamos ahí y lo utilizamos en lugar de copiarlo a la memoria de sistema para después cargarlo a la memoria de la tarjeta gráfica.

Capítulo 4

Mejora de la aplicación

Para la realización de este proyecto fin de carrera se decidió buscar herramientas que hicieran uso de tarjetas gráficas para comprobar contraseñas. Tras una búsqueda exhaustiva se determina que la aplicación *Distributed Hash Cracker* era la más completa y, lo más importante, la única que disponía de una licencia abierta que nos permitía poder realizar cambios sobre ella.

Como ya se mencionó en la introducción, el uso de software libre es muy importante para la seguridad por el hecho de que al disponer del código fuente se pueden realizar auditorías sobre el mismo. Además, en caso de que la empresa que desarrolla el software dejase de darle soporte siempre cabría la posibilidad de que otra empresa pudiera ofrecer dicho soporte. Por otra parte, el software libre no solo tiene ventajas en la seguridad, sino que ofrece la garantía de que si la empresa deja de desarrollar dicho software, o ésta desapareciese, siempre quedaría el software a disposición del público pudiendo tomar otro el relevo sobre el mantenimiento, sin que puedan realizarse acciones legales contra éste por infracción de derechos de autor. En este punto se debe mencionar que hay que obedecer siempre los términos de la licencia del software y, en especial, el Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el Texto Refundido de la Ley de Propiedad Intelectual, que es la ley que regula los derechos de autor en España.

Distributed Hash Cracker, en adelante DHC, es un software que permite determinar la palabra utilizada en una función resumen para devolver un determinado resumen. Para ello hace uso de un sistema de fuerza bruta [Zon09]. Éste software se encontraba alojado en la página `http://rpisec.net/`, pero por cuestiones desconocidas ha desaparecido dejando de tener soporte. Esto

nos ha animado a realizar las modificaciones que necesitábamos para conseguir una aplicación completa y, especialmente, más fácil de mantener y que permitiera añadir funcionalidades de forma sencilla.

DHC es un software que se distribuía bajo licencia BSD[1]. Esta herramienta está programada en C++, ensamblador y PHP y es capaz de distribuir el trabajo entre un conjunto de máquinas, denominadas agentes. Los agentes se encargan de procesar tareas criptográficas tanto en CPU como en GPU dependiendo de los algoritmos implementados en cada caso.

Aunque DHC es una muy buena herramienta tiene algunas deficiencias que hacen que necesite modificaciones en algunos puntos importantes de su código para garantizar que la herramienta pueda sobrevivir en el futuro.

El mayor problema actualmente es que aún estando escrito en un lenguaje orientado a objetos, no se hace uso de las características del mismo cuando se debería. Por ejemplo, el código de las funciones resumen de que dispone está entremezclado. Esto supone un gran problema al implementar nuevas funciones resumen, ya que implica tener que ver qué partes de código se deben adaptar y cuáles no. Por tanto, es necesario hacer un cambio de cómo controla DHC estas funciones de tal modo que sea más sencillo el mantenimiento de la herramienta.

Por otra parte, la implementación de funciones resumen de que dispone DHC, aún estando relativamente bien surtida, creemos que podría ser necesario en un futuro la implementación de más funciones resumen u otro tipo de comprobaciones sobre seguridad (como pudiera ser redes WiFi). Esto significa que hay que realizar los cambios necesarios que permitan en un futuro añadir funcionalidades diferentes a las inicialmente planteadas sin que esto suponga un esfuerzo demasiado elevado.

Finalmente, DHC dispone de un programa que se encarga de controlar el reparto de las tareas. Este programa se denomina controlador y está desarrollado enteramente en PHP sin seguir un modelo claro de organización del código. En un principio parece que está basado en el patrón arquitectónico MVC, pero por algún motivo el código fuente parece muy entremezclado.

[1]La licencia libre BSD permite realizar modificaciones sobre el software siempre y cuando se mantenga la mención de autoría en el mismo, pero no obliga a distribuir el código en caso de dar copias de la aplicación.

4.1. Lenguajes de programación utilizados

El lenguaje de programación elegido para el proyecto ha sido C++. Esta elección se realiza por diversos factores:

- Tanto los lenguajes C como C++ son completamente compatibles con CUDA. Además, CUDA ofrece algunas ayudas específicas para C++ (como la forma de definir texturas) facilitando el trabajo con dicha tecnología. Éste tipo de ayudas es más patente cuando se utiliza el método sencillo de desarrollo.
- El autor del presente proyecto final de carrera está muy familiarizado con dicho lenguaje lo que permite realizar mejor el desarrollo.
- Aunque no hace un uso adecuado de las capacidades de C++, DHC ha sido desarrollado utilizando dicho lenguaje de programación.

Por otra parte se hace uso del lenguaje PHP para el controlador web por ser un lenguaje gratuito y muy extendido. Además, dispone de versiones para Windows, Linux y MacOS X.

Por último se hace uso de una variación del lenguaje C creada por NVIDIA para desarrollar en sus tarjetas gráficas. Esta variación consiste en añadir información sobre el acceso a métodos y variables:

- Se usa __global__ para denotar funciones que se alojarán en la tarjeta gráfica, pero que podrán ser llamadas desde la CPU.
- __device__ son funciones que solo pueden ser llamadas desde otras funciones que ya se encuentren en la GPU.
- Las variables que pueden ser compartidas entre distintos *threads* se denotan con __shared__. Estas variables hacen uso de la memoria global.
- Si se necesitase que una función resida en la CPU se le indicaría con __host__.
- Si utilizamos memoria que no pueda cambiar de valor dentro de la GPU utilizaremos la notación __constant__ y deberemos iniciar la memoria desde la CPU.

En caso de necesitar más detalles sobre la arquitectura CUDA en [NVI10] se puede encontrar una guía completa sobre las peculiaridades de la implementación de C realizada por NVIDIA.

4.2. Estudio de DHC

DHC es una aplicación divida en dos partes claramente diferenciadas y que es importante comprender de cara a poder realizar modificaciones de forma satisfactorias sobre el. Estas partes son el controlador y el agente.

4.2.1. Controlador de DHC

El controlador es la herramienta encargada de gestionar las tareas solicitadas, permitir al usuario la introducción de tareas y la cancelación de las mismas, realizar estadísticas de uso, etc. A continuación se expone de manera más detallada las tareas realizadas por el controlador:

- Introducción de nuevos resúmenes a probar. Estos resúmenes son almacenados en una base de datos para garantizar la persistencia de la tareas y así, en caso de caída, poder recuperar el trabajo.
- Crear paquetes de claves a probar y repartirlos entre los agentes para comprobar un determinado resumen. Estos paquetes son denominados unidades de trabajo (WU en sus siglas en inglés y que será la nomenclatura utilizada en este documento). Una WU es una estructura que contiene la información de la tarea a realizar. Esta información contiene datos como:
    - Tipo de algoritmo que se desea utilizar, como pueden ser MD5, SHA-1, etc.
    - Datos sobre los que realizar las comprobaciones. En el caso de las funciones resumen este dato se corresponde con el hash que queremos comprobar.
    - Juego de caracteres a utilizar. Permite elegir si utilizar número y letras minúsculas o mayúsculas. También permite seleccionar símbolos, espacio y retorno de carro.
    - Longitud máxima esperada. Ésta se utiliza para restringir las pruebas, ya que consideramos que una cotraseña no va a exceder de los 8 caracteres no es necesario probar más allá.
    - Bloque de claves a comprobar. Esto es el conjunto de contraseñas que se van a probar indicadas por su posición inicial (el primer valor a probar) y final.

- Mide los tiempos transcurridos desde que un controlador recibe una WU hasta que devuelve un resultado, calcula la velocidad de ejecución en hash/s y muestra informes de velocidad y carga de trabajo del sistema.

- Controlar los tiempos de expiración de las tareas y las prioridades de las mismas.

De este modo el controlador es una de las partes más importantes del sistema al ser la interfaz que el usuario va a utilizar para utilizar los recursos del mismo.

El controlador hace uso de una base de datos MySQL, conocida por ofrecer un buen rendimiento y por ser software libre. Gracias al uso de esta base de datos la aplicación consigue de forma sencilla garantizar la persistencia de los datos en caso de caídas del sistema. Hay que recordar que sólo hay un controlador y en caso de que éste deje de estar activo todo el sistema se viene abajo, por lo que es importante garantizar que la información con la que éste trabaja no se pierda.

Cada vez que una WU es dada a un agente, se calcula un tiempo de vida que será el tiempo que tiene el agente para devolver un resultado antes de considerar que esa WU ha caducado. Si una WU caduca será reciclada y ofrecida a otro agente.

El controlador esta programado enteramente en PHP sin hacer uso de ningún *framework* sino que haciendo uso del patrón arquitectónico MVC se ha programado desde cero todo el sistema. Al no disponer de documentación se hace relativamente complejo realizar el mantenimiento de esta solución, especialmente cuando no siempre hace uso del patrón anteriormente mencionado.

4.2.2. Agente de DHC

El agente es el encargado de realizar la tarea más compleja del sistema que es la comprobación de los resúmenes dados por el controlador utilizando para ello las herramientas que estén a su disposición (la CPU y si puede ser la GPU).

El agente se encuentra en el equipo que va a realizar las operaciones de comprobación de funciones resúmenes y puede haber tantos equipos como se necesite. De este modo el sistema no está restringido a un único agente

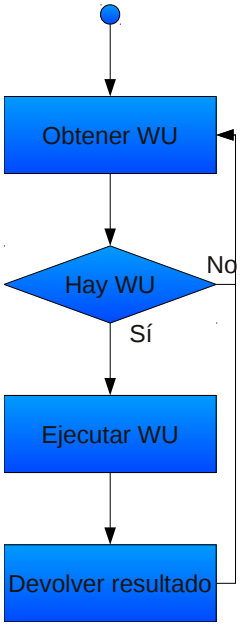

Figura 4.1: *Proceso de ejecución de un controlador*

y puede distribuir la carga de trabajo entre tantos agentes como haga falta. Además, los agentes no tienen porque estar restringidos sólo a una red local, también pueden distribuirse a lo largo de internet, de modo se podría tener varias sedes que ejecuten agentes disponiendo de un único controlador. En caso de requerir un sistema más complejo se puede consultar el apartado C.4.

Por otro lado, cuando se ejecuta un agente sobre un ordenador, éste realiza una comprobación del número de CPUs en el sistema y del número de GPUs con capacidad para utilizar CUDA. Lo que se pretende con esto es crear un proceso ligero encargado de cada unidad de cómputo que haya en el sistema, teniendo en cuenta que para cada GPU asigna una CPU. Esto significa que si disponemos de un sistema con 4 GPU y 8 CPUs (concretamente 8 núcleos) sólo considerará que hay 4 GPUs y 4 CPUs (las otras 4 CPUs las utiliza para control de las unidades de tarjeta gráfica).

A grandes rasgos, el proceso de ejecución de un agente, tras haberse configurado es el mostrado en la figura 4.1. Los apartados más importantes de este proceso son la obtención de la WU y la ejecución de la misma.

Durante el proceso de obtención de una WU, el agente comprueba si esta es valida para el algoritmo indicado. Para ello utiliza el siguiente código:

```
if(algorithm == "md4" ||
   algorithm == "md4_fast" ||
   algorithm == "md5" ||
   algorithm == "md5crypt" ||
   algorithm == "ntlm")
{
   hashlen = 16;
}
else if(algorithm == "sha1")
   hashlen = 20;
else
   ThrowCustomError("Unknown hash function");

if(algorithm != "md5crypt" && hashlen*2 != text.length())
   ThrowError("Invalid hash length");

//Sanity check
if(hashlen > 256)
   ThrowError("Hash length too long");
if(text.length() == 0)
   ThrowError("Empty hash");
```

El problema del código mostrado anteriormente es que requiere forzosamente ser modificado ante cualquier cambio de un algoritmo, o ante la creación de un algoritmo nuevo. De este modo el código no aprovecha la capacidad de abstracción que otorga el lenguaje C++, ya que podría haberse hecho esto de un modo más elegante como se comprobará más adelante. Además, el código mostrado supone un problema para el mantenimiento ya que éste se encuentra en un fichero del código fuente, pero además, en otro completamente distinto también se hacen comprobaciones sobre los algoritmos por lo que hay que tener un buen conocimiento sobre la estructura interna de los ficheros del programa.

4.3. Modificaciones realizadas a DHC

Como se ha podido ver en el apartado anterior DHC es una buena aplicación, pero tiene algunos fallos que podrían resolverse para mejorar la mantenibilidad de la solución. Esto ofrecería una mayor esperanza de vida al

programa y lo convertiría en una herramienta mucho más útil a corto y largo plazo.

Por los motivos anteriores se ha decidido realizar los cambios necesarios que conviertan a DHC en una herramienta más fácil de manejar, de mantener y que permita a cualquier programador añadir funcionalidades de forma sencilla sin requerir un alto grado de conocimiento del código del agente.

4.3.1. Diseño de API para algoritmos de codificación

Como se ha comentado anteriormente, una de las mayores deficiencias de DHC, a nuestro entender, es la falta de un mecanismo sencillo para incorporar nuevos algoritmos debido, principalmente, a que el código de los algoritmos se entremezcla con la funcionalidad del controlador en distintos ficheros de código fuente. Esto supone un problema importante, a la hora de realizar nuevos algoritmos, al encontrarse todo el código de los mismos repartido entre distintas funciones y métodos, dificultando enormemente las labores de mantenimiento y de desarrollo. Por este motivo se ha decidido implementar una API para facilitar el desarrollo de algoritmos, que aprovechando todas las funcionalidades ya existentes, mejore de forma sustancial las labores de mantenimiento de la aplicación.

El diseño de la API propuesta se ha basado en dos ideas:

- El algoritmo como sistema para la realización de una tarea específica, como pueda ser obtener el valor que generó un determinado resumen y
- La factoría, que es el mecanismo que permite acceder a los algoritmos que existan en el sistema.

El objetivo principal de la propuesta es permitir agrupar todo el código que rige a un algoritmo de forma que facilite la modificación del mismo y, en caso necesario, crear algoritmos nuevos sin tener que realizar un gran esfuerzo.

Gracias al API propuesto, cada algoritmo será una clase que herede de *Algorithm*. Esta clase abstracta define las funcionalidades mínimas que debe implementar un algoritmo para poder ser utilizado por el agente. Así se consigue englobar en un único lugar todo lo relativo a un algoritmo.

La clase *Algorithm* define los siguiente métodos obligatorios de implementar:

GetName Devuelve el nombre del algoritmo. Este nombre se utiliza para poder identificar el algoritmo y poder seleccionarlo cuando se solicite. Para garantizar el buen funcionamiento del sistema y evitar funcionamientos extraños, este nombre debe ser único.

HashLength Es el tamaño que debe tener un hash para ser considerado válido. Con esto se pretende que en caso de que el controlador nos de un hash erróneo poder hacer frente al problema.

InputLength Tamaño de la entrada recibida por el WU. Aunque en muchos casos solo comprobando el tamaño del hash sería suficiente, hay algoritmos que pueden necesitar entradas extendidas, por lo que se necesita un doble chequeo.

ExecuteCPU Implementa el código que se encarga de utilizar la CPU para realizar la tarea solicitada.

ExecuteGPU Ejecuta en una GPU el algoritmo con objeto de realizar la tarea de comprobación.

IsGPUCapable Indica si el algoritmo tiene soporte de GPU. De este modo el sistema sabe si puede determinar el método por el que debe realizarse el procesamiento, si por CPU o por GPU.

IsCPUCapable Indica si el algoritmo soporta procesamiento utilizando CPU.

Por otra parte se ha creado una factoría de algoritmos (la clase *AlgorithmFactory*) que permite acceder de forma sencilla a los algoritmos que se encuentren registrados en el sistema. Además, esta factoría permite el registro de nuevos algoritmos de forma sencilla para facilitar la implementación de nuevas utilidades.

Para poder comprender mejor como funciona el nuevo sistema la figura 4.2 muestra a grandes rasgos como sería el proceso. En este caso el actor es el agente que va a hacer uso de la funcionalidad ofrecida para realizar una comprobación sobre MD5.

A la hora de implementar estas características hay que tener en cuenta que se puede hacer uso tanto de una CPU como de una GPU. Esto supone que el algoritmo debe estar capacitado para ofrecer alguna de dichas opciones y, a poder ser, ambas. Esto supone en la práctica que tanto el algoritmo

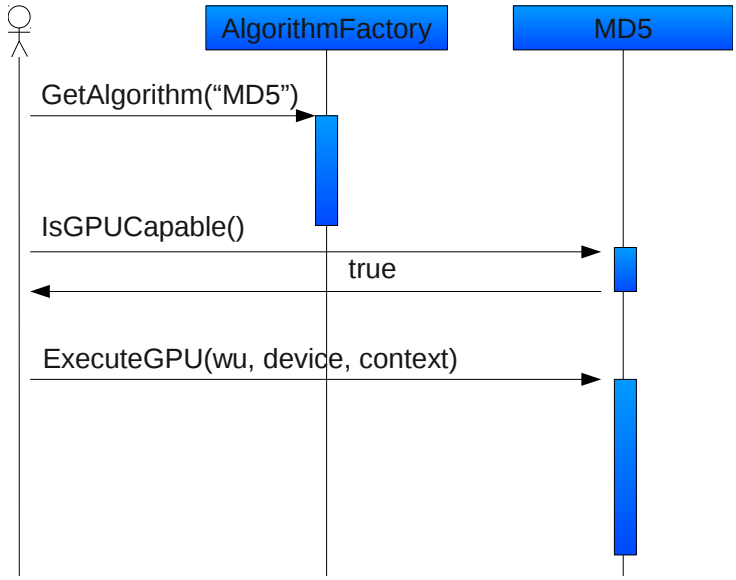

Figura 4.2: *Diagrama de secuencia de la ejecución de un algoritmo en GPU*

implementado puede esta duplicado disponiendo del código de la versión de CPU y de la versión de GPU.

4.3.2. Diseño de API para la estandarización de la ejecución de algoritmos

Una vez creado el API para algoritmos se puede comprobar que muchos algoritmos siempre siguen el mismo patrón de ejecución. Esto supone tener que repetir constantemente el mismo código en los algoritmos cuando podríamos hacer uso de la reutilización para eliminar este inconveniente. Por este motivo se planteó diseñar un sistema que permitiera estandarizar la forma que tiene un algoritmo de ejecutarse de tal modo que sólo tengamos que centrarnos en lo imprescindible de cada uno de ellos.

La filosofía que se ha empleado es la misma que en el caso de los algoritmos. En este caso se ha creado la clase *Executor* que se encarga de definir las funcionalidades básicas que debe implementar un mecanismo de ejecución y por otro lado se dispone de la clase *ExecutorFactory* que nos facilita el acceso a los mismos.

Con este sistema se obtiene un mecanismo muy interesante para la realización de pruebas de algoritmos sin tener que modificar el resto de algoritmos

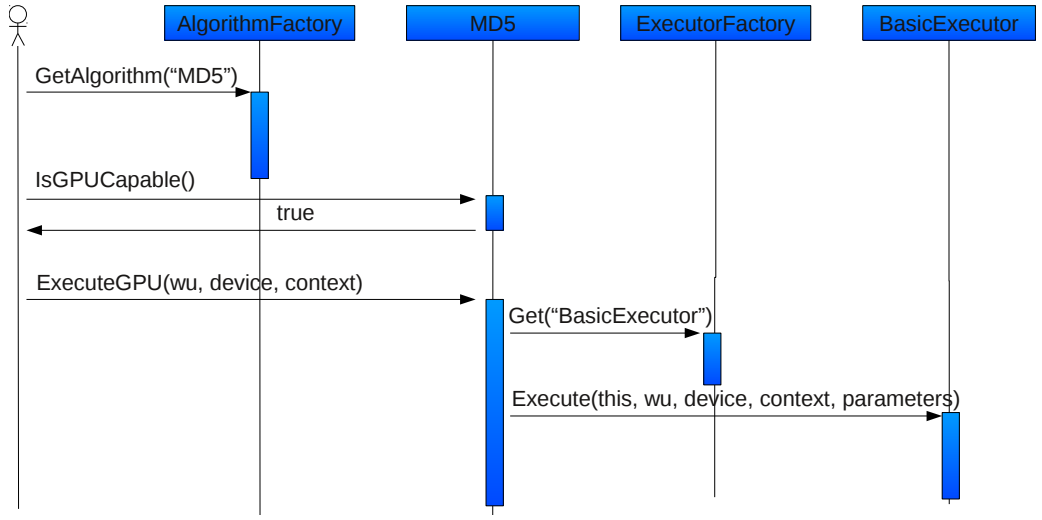

Figura 4.3: *Diagrama de secuencia de la ejecución de un algoritmo en GPU utilizando un* Executor

y, en caso de obtener un mecanismo suficientemente bueno y que pueda ser utilizado por más algoritmos, de organizar todos los algoritmos que deban ejecutarse de igual forma para reducir la cantidad de código duplicado facilitando el mantenimiento en caso de errores.

El código original de DHC utiliza el mismo mecanismo de ejecución para todos los algoritmos implementados. Esto está muy bien hasta cierto punto ya que restringe de forma muy clara la cantidad de mecanismos que pueden implementarse en el sistema. Gracias a la nueva API se permite implementar más algoritmos, no teniendo que restringirnos necesariamente a una forma concreta de hacer las cosas o a un tipo concreto de algoritmos (hasta el momento solo hay funciones resumen).

La ejecución con este mecanismo se complica un poco más (ver figura 4.3), pero la sobrecarga introducida por este mecanismo es despreciable en tiempo. Además, hay que tener en cuenta que no es obligatorio el uso de un *Executor* por parte de un algoritmo.

Una de las principales ventajas que podemos comentar sobre este sistema es que ahora un algoritmo no tiene porque estar sujeto a un único modo de hacer las cosas. Se puede detectar si hay un *Executor* en concreto y si no está se pude seleccionar otro. Así, en caso de que se retire uno por cuestiones de mantenimiento o depuración el sistema podría seguir funcionando (siempre que se implemente esta forma de trabajar).

Por otra parte se permite, gracias a la arquitectura utilizada, la creación

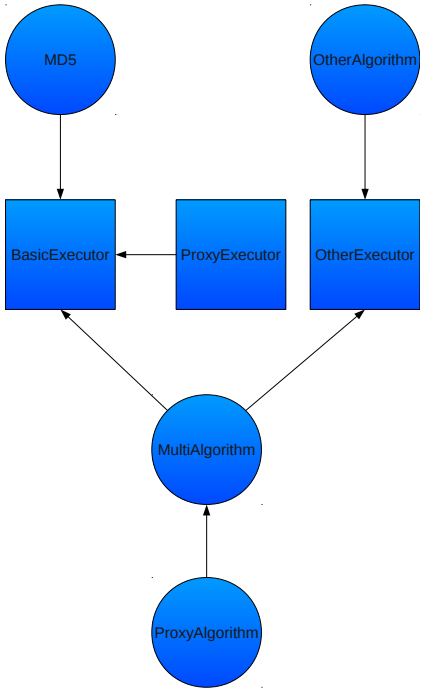

Figura 4.4: *Panorámica de uso de* Algorithms *y* Executors

de *proxys*, objetos intermedios que capturan la funcionalidad expuesta para realizar operaciones de forma transparente. Por ejemplo, se podría disponer de un *proxy* para los algoritmos que permitiese hacer un seguimiento de las llamadas ejecutadas en caso de que el algoritmo no tuviese habilitada la generación de trazas de ejecución.

En la figura 4.4 puede verse como pueden interactuar entre sí los distintos elementos expuestos en este apartado. Como puede comprobarse, la versatilidad del nuevo sistema permite gran cantidad de configuraciones que pueden ser utilizadas siempre que sea necesario. Para comprender mejor la gráfica, los círculos representan algoritmos, las cajas son *Executors* y las flechas son asociaciones de uso. Por ejemplo, *MD5* usa *BasicExecutor*.

4.3.3. Modificaciones para la mejora de la escalabilidad

Durante las primeras pruebas realizadas sobre DHC se notó que el tiempo de respuesta de la interfaz web podía verse seriamente afectado si la velocidad

utilizada por los agentes para solicitar tareas o devolver tareas es muy alta y si la distancia entre el agente y el controlador es grande. Este último punto se constató cuando se hicieron pruebas en las que el controlador se encontraba en un servidor en Alemania y el agente se encontraba en España. Sin duda la ralentización en este punto se debía a las latencias de la red por la espera de los asentimientos (paquetes ACK de la arquitectura IP). Estos retrasos pueden afectar de forma importante en la escalabilidad del sistema, por lo que se ha decidido realizar cambios para eliminar en la medida de lo posible estos problemas.

Cada agente de DHC se encuentra en un bucle en el que en cada iteración solicita una WU al controlador. En caso de que no haya ninguna tarea que pueda realizar, el agente procede a esperar 5 segundos. Este tiempo puede resultar insuficiente si el controlador tiene que hacer gran cantidad de cálculos o si el número de agentes es tal que saturen con peticiones al controlador (efecto DDoS). Por este motivo se ha diseñado un algoritmo que ajuste el tiempo de espera de forma dinámica para adaptarse mejor a las capacidades del controlador.

El nuevo algoritmo tiene en cuenta el tiempo que tarda el controlador en dar una respuesta al agente y si la respuesta dada contiene o no una WU. Con esto se pretende que el sistema se ajuste mejor a las condiciones ambientales del sistema.

Control del tiempo con respecto al tiempo de respuesta del controlador

Siempre que se solicitan WU a un controlador el agente mide los tiempos que tarda en recibir la respuesta. Este tiempo permite controlar si el controlador está empezando a saturarse o no. Así, en caso de que los tiempos empiecen a verse incrementados, y por tanto se considere que el controlador está saturado, se procede a incrementar el tiempo entre solicitudes. Con esto se pretende descargar al controlador de un exceso de carga distribuyéndola en el tiempo.

Por otra parte, en caso de que el tiempo disminuya supondría justo lo contrario, la carga del controlador es menor, por lo que el tiempo de espera puede empezar a reducirse poco a poco. Al realizar una disminución lenta del tiempo de espera se consigue que no se sature de repente el controlador como podría suceder si este tiempo se redujera rápidamente.

Control del tiempo con respecto a las tareas recibidas

Aunque el ideal sería que el sistema pudiera estar constantemente funcionando, la realidad dista mucho de esto ya que es fácil que se de el caso de que no haya tareas que procesar o, que habiéndolas, éstas no sean compatibles con los agentes libres.

Teniendo lo anterior en cuenta, es fácil considerar que si en un momento dado no hay tareas, la próxima vez que consultemos al agente éste siga sin disponer de trabajos que asignar. Como este caso puede darse de forma continuada durante muchas consultas creemos que tiene sentido considerar que cada vez que se haga una petición a un controlador, y éste no tenga WUs, la espera hasta la siguiente vez que solicite una tarea se vea incrementado hasta un tiempo máximo.

En caso de que el controlador empiece a ofrecer de nuevo WU a los agentes, estos empezarán a disminuir el tiempo entre solicitudes de forma paulatina hasta que vuelvan a estar estables.

En la figura 4.5 puede verse una explicación gráfica de los dos mecanismos explicados. Las variables utilizadas son:

Tesp: representa el tiempo de espera que se utilizará en caso de que se considere que el sistema no está respondiendo correctamente.

Tresp: contiene el tiempo que ha tardado el controlador en devolver una respuesta al agente.

Tant: es el tiempo que tardó el agente en responder en la anterior consulta.

MAX_TESP: representa el tiempo máximo entre consultas al controlador.

MIN_TESP: es el tiempo mínimo entre llamadas al controlador.

4.4. Diseño e implementación de un mecanismo de carga de extensiones

Ahora que se dispone de un API que facilita la creación de algoritmos y mecanismos para ejecutar estos, se ha diseñado un sistema que permite cargar nuevos algoritmos y *Executors* que se encuentren fuera de la aplicación. Estos

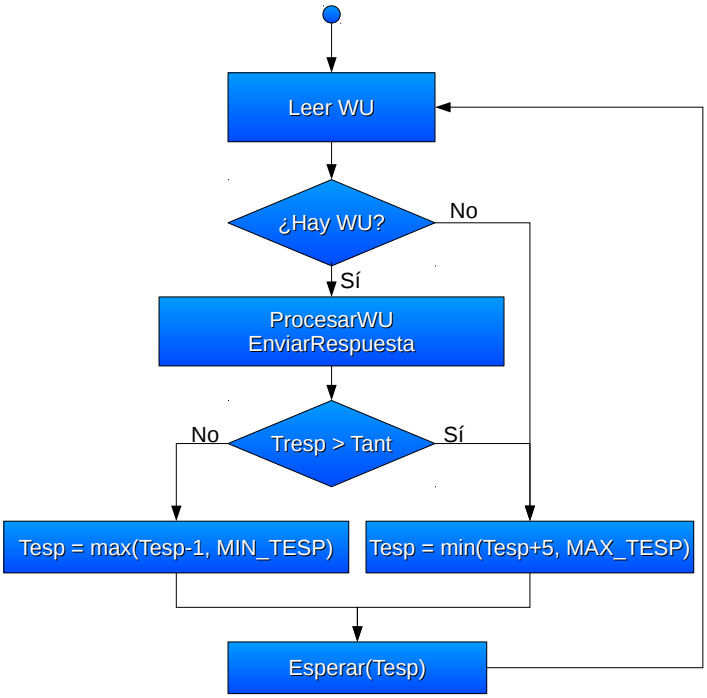

Figura 4.5: *Control del tiempo de espera en las solicitudes*

algoritmos y *Executors* que pueden cargarse de este modo son denominados *plugins*.

Para comprender mejor la idea hay que tener en cuenta que hasta este momento, si se modificaba un algoritmo al estar éste dentro del programa había que generar un nuevo ejecutable con dicha modificación. Lo que se pretende es que el algoritmo no tenga que formar parte del agente y que pueda ser una unidad que se encuentre por separado.

Para poder implementar este apartado se ha desarrollado una serie de clases, (figura 4.6) que van a permitir describir el contenido del mismo y añaden facilidades para poder crear instancias de las clases que defina, que se presentan a continuación:

PluginFacility: clase abstracta que describe una funcionalidad que va a presentar el *plugin* al agente. Ésta está identificada por un nombre, su versión y el tipo. Además, permite crear instancias de la funcionalidad.

PluginFacilityAlgorithm: permite que el *plugin* pueda ofrecer algoritmos a los agentes. Esta clase implementa PluginFacility.

PluginFacilityExecutor: permite que el *plugin* pueda ofrecer *Executors* a los agentes. Esta clase implementa PluginFacility.

PluginFactory: esta clase debe ser instanciada en el *plugin* y en ella hay que registrar todos aquellos componentes que ofrecerá el *plugin* al agente. Cada componente a ofrecer deberá heredar de PluginFacility. Además, esta clase proporciona información extra como el autor del *plugin* o el listado de *facilites* que se van a exportar.

Además de lo anterior, los *plugins* deben contener una función llamda *GetPluginFactory* que permitirá obtener la instancia de *PluginFactory* que tiene el *plugin*.

Como tener que memorizar todo el proceso de crear un *plugin* puede resultar muy complicado (intervienen muchas clases con dependencias entre ellas) se ha añadido una serie de macros de C que permiten simplificar en gran medida la creación de los mismos. Por ejemplo, para la creación del algoritmo que implementa SHA-256 se ha creado una clase denominada sha256 que exportamos como algoritmo del siguiente modo:

```
BEGIN_PLUGIN("Samuel Rodriguez Sevilla")
```

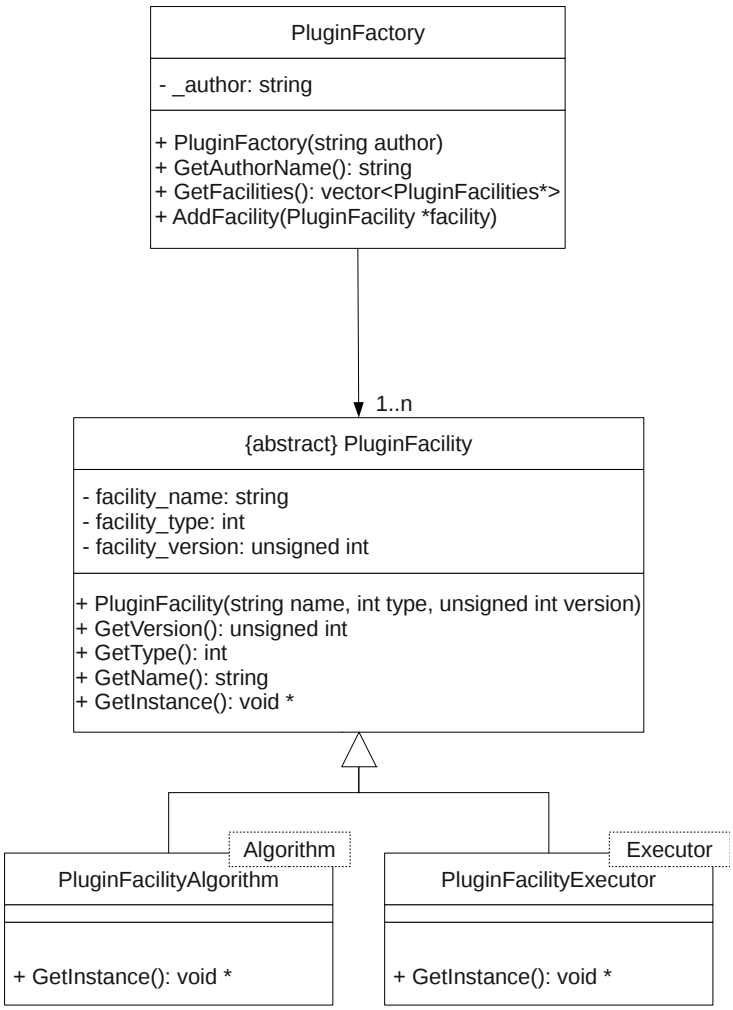

Figura 4.6: *Diagrama de clases del sistema de plugins*

```
   ADD_ALGORITHM(sha256, "sha256", 1)
END_PLUGIN()
```

Las macros existentes son:

BEGIN_PLUGIN(author_name) Indica que se va a definir un *plugin* y su autor es *author_name*.

END_PLUGIN() Finaliza la definición de un *plugin*.

ADD_ALGORITHM(class, name, verion) Añade un algoritmo definido en la clase *class* que será identificado por *name* y cuya versión es *version*.

ADD_EXECUTOR(class, name, verion) Igual que el algoritmo, pero para un *Executor*.

4.5. Mejoras en la depuración

Cuando se desarrollan aplicaciones siempre hay que tener muy en cuenta que se pueden producir errores durante la ejecución. Generalmente, el método habitual para comprobar los fallos es utilizar salidas por pantalla tratando de acotar el lugar del fallo; este sistema es poco profesional por lo que para la realización de esta práctica se han utilizado otros mecanismos.

Para comprobar la ejecución de toda aplicación en caso de errores el mejor método es el uso de un depurador, por este motivo se ha utilizado el depurador GNU GDB. Pero aún así no siempre es suficiente con utilizar un buen depurador, sino que tener información extra a la que éste pueda darnos puede ser de gran utilidad. Por este motivo se ha implementado un sistema de trazas para depuración que no buscan la acotación de los errores, esa información ya se encuentra en el fichero de volcado creado en los fallos graves, sino que ofrece una visión más global sobre qué está sucediendo en el código. Esta información puede ser desde cuando se entra en un método, cuando se sale de éste, salida de información varia, etc.

Para activar la salida de depuración se debe llamar a CMake del siguiente modo:

```
$ DEBUG=1 cmake .
```

Así, cuando CMake esté generando el fichero de Makefile tendrá conocimiento de que debe activar la opción de depuración y de generación de salida de depuración. El primero consiste en añadir el parámetro -g durante la compilación y el segundo con -DDEBUG.

Las funciones para utilizar la salida de depuración son:

DO_ENTER(class, method) Se encarga de anunciar cuándo se entra en un método/función. También configura automáticamente el sistema para controlar cuándo se sale del método/función para que quede constancia. Sus parámetros son cadenas que representan la clase en la que nos encontramos (*class*) y el método (*method*).

DO_ERROR(str) Muestra un error por la salida de depuración. Acepta como parámetro la salida que debe mostrar.

DO_WARNING(str) Advierte de una situación extraña, pero que no impide a la aplicación seguir funcionando. Acepta como parámetro la salida que debe mostrar.

DO_MESSAGE(str) Muestra un mensaje de usuario. Acepta como parámetro la salida que debe mostrar.

DO_LOG(str) Muestra un mensaje de seguimiento. Acepta como parámetro la salida que debe mostrar.

DO_DEBUG(str) Información de depuración. Es útil cuando se quiere mostrar por la salida de depuración información sobre el estado de ciertas variables. Acepta como parámetro la salida que debe mostrar.

INIT_LOG(lvl) Inicializa el sistema de salida de depuración. Acepta como parámetro el nivel de depuración que se va a utilizar. Este nivel puede ser:

- **ERROR** Es el nivel más bajo de depuración. Solo se muestran las salidas de error.
- **WARINING** Siguiente nivel de depuración. Se mostrarán las salidas de los niveles anteriores y las salidas de avisos.
- **MESSAGE** Muestra las salidas de los niveles anteriores y los mensajes del programador.

LOG Mensajes de seguimiento y todos los mensajes de los niveles anteriores.

DEBUG Mensajes de depuración junto a todos los mensajes de los niveles anteriores.

4.6. Nuevo controlador

Además de los cambios anteriores realizados al agente también se han realizado cambios en el controlador. Concretamente se ha rehecho completamente reutilizando el código justo. El motivo de la creación de este nuevo controlador se debe, principalmente, a que se quiere disponer de un sistema bien organizado y claramente centrado en la labor de controlar las tareas.

Hasta este momento el controlador se encargaba tanto de administrar las tareas como de codificar los mecanismos necesarios de control para determinar los datos de entrada, las funciones llamadas, etc. Esto se debe principalmente a que no se ha hecho uso de ningún *framework* de desarrollo web que ocultase este tipo de tareas. De este modo el controlador es una aplicación con un código que entremezcla la representación de datos, el control del flujo de ejecución y las tareas administrativas.

Por el motivo anterior se decidió crear un nuevo controlador haciendo uso del *framework* CakePHP (puede encontrarse en `http://cakephp.org/`). Este *framework* tiene las siguientes características:

- Abstrae la base de datos facilitando el trabajo con los datos independientemente del motor de base de datos utilizado.
- Maneja el flujo de funcionamiento de la aplicación permitiendo, de este modo, que el programador se centre únicamente en la labor de desarrollo de las funcionalidades del programa.
- Tiene actualizaciones constantes que, en caso de mejoras de rendimiento o estabilidad, pueden incorporarse fácilmente a nuestro desarrollo sin necesidad de realizar cambios sobre el mismo.
- Dispone de abundante documentación con gran cantidad de ejemplo, lo que facilita su aprendizaje.
- Su uso es muy sencillo.

Por otra parte, aprovechando este cambio del controlador se ha intentado realizar algunas optimizaciones. Entre ellas podemos destacar:

- Optimización de la base de datos. Para ello se estudió el uso que hace el controlador de los accesos y se comprobó que las tablas que se utilizan para realizar las estadísticas son constantemente utilizadas y estas nunca llegan a tener un número de elementos muy elevado (el controlador borra los datos que considera antiguos). Por este motivo se ha decidido pasar estas tablas a la memoria RAM, de modo que el acceso sea más rápido. El mayor problema de esta solución es que en caso de caída de la base de datos se perderían estos valores, pero al no ser información fundamental no supone una gran preocupación.

- Agrupación de accesos a base de datos. En el antiguo controlador se podía encontrar acciones que realizaban dos o tres accesos a la base de datos para modificar un mismo elemento. Para evitar esta situación lo que se ha hecho ha sido ir acumulando los cambios a realizar y una vez que no se van a hacer más se hace la actualización de la base de datos. Esto permite ahorrar algo de tiempo y reduce la carga sobre la base de datos.

Se ha tenido en cuenta el diseño de la web a la hora de reescribir el controlador considerando que un estilo más atractivo ayudaría a una mejor experiencia de usuario.

4.7. Algoritmos nuevos de seguridad implementados

Inicialmente DHC soporta los algoritmos MD4, MD5, SHA1, NTLM y MD5 crypt. Este último es una versión de MD5 utilizada por los sistemas UNIX para almacenar las contraseñas y que se forma de la siguiente forma IDSALT$HASH, donde ID es el identificador del algoritmo de resumen empleado (1 para MD5), SALT es una semilla aleatoria que se adjunta a la contraseña a resumir y HASH es el resultado del resumen de la unión de la contraseña y la semilla:

$$HASH = H(SALT|PASSWD)$$

Estos algoritmos se han recodificado para que se adapten a los cambios mencionados en 4.3.1. Tras este cambio se ha procedido al desarrollo de nuevos algoritmos que dotan al sistema de mayor funcionalidad, especialmente si son algoritmos más modernos.

Hay que tener en cuenta que a menos que se hallen debilidades contra los algoritmos el sistema de comprobación a utilizar será siempre el de fuerza bruta. Esto supone que los algoritmos más actuales sean más costosos con respecto al tiempo necesario.

Para el desarrollo de los nuevos algoritmos se ha tenido en cuenta su uso. De este modo no se han implementado aquellos que están en desuso por ser de poca utilidad.

4.7.1. SHA-256

Este algoritmo de resumen busca sustituir al antiguo SHA-1 ya que este se vio seriamente afectado por el tamaño de los resúmenes que generaba (de 128 bits) y por el descubrimiento de ataques controla el mismo (como se ha visto en 2.1.3).

Para codificar este nuevo algoritmo se ha procurado hacer uso de la experiencia previa de la aplicación, procedente del estudio de la misma, y se ha reutilizado parte del código del antiguo SHA-1. De este modo se consigue reducir el tiempo de desarrollo.

La implementación de este algoritmo se ha tomado a partir de la información ofrecida por el NIST [NIS08] y se ha probado que funciona correctamente.

Capítulo 5

Resultados

En este capítulo se presentan las pruebas realizadas sobre el proyecto y los resultados obtenidos por las mismas.

Con las pruebas realizadas se ha procurado asegurar el buen funcionamiento de los cambios introducidos durante la realización del proyecto para garantizar su utilidad.

5.1. Subsistema de *plugins*

Las pruebas sobre el sistema de *plugins* tratan de comprobar, por una parte, el buen funcionamiento de la carga dinámica de las extensiones, así como de las mejoras suministradas por el API de algoritmos y de *executors*.

Este apartado ha sido uno de los más importantes ya que todos los cambios realizados tienen un gran impacto sobre la estructura del proyecto. De este modo se debe garantizar que tras los cambios todo el sistema sigue funcionando de forma correcta.

Los pasos seguidos para el propósito descrito han sido el siguiente:

1. Paso del algoritmo de MD4 al nuevo sistema.

   Con esto se pretendía probar si el mecanismo de control de algoritmos funcionaba correctamente sin influir en los resultados.

   Se eligió el algoritmo de MD4 al azar entre todos los implementados en DHC.

2. Paso de todos los algoritmos antiguos al nuevo sistema.

Así se garantizaba el buen funcionamiento en todos los casos y se podía comprobar que no hubiese código antiguo interfiriendo en en la ejecución.

3. Creación de *BasicExecutor*.

   Este *executor* implementa la lógica antigua de funcionamiento de los algoritmos. Se pretende de este modo reducir la cantidad de código duplicado y comprobar a su vez el funcionamiento de este subsistema.

   Como en el caso anterior el primer algoritmo en hacer uso de esta funcionalidad ha sido MD4.

4. Uso de *BasicExecutor* en todos los algoritmos.

   Con este cambio se comprueba el funcionamiento real de todos los algoritmos del sistema.

5. Creación del plugin *DummyPlug*.

   Este plugin de prueba no realiza ninguna operación dentro del sistema, pero es útil para comprobar que el subsistema de *plugins* es capaz de cargar correctamente los mismos.

6. Transformación de los algoritmos en *plugins*.

   Finalmente, se pasaron todos los algoritmos para que hagan uso del sistema de *plugins* y así comprobar el funcionamiento real; de este modo se independizan los *plugins* del agente.

Para realizar las pruebas anteriores se fueron realizando los cambios paulatinamente y corrigiendo los errores que surgieron. En estos momentos se puede considerar que el subsistema de *plugins*, algoritmos y *executors* funciona correctamente.

5.2. Controlador

El controlador ha sufrido un rediseño importante de su aspecto que ha buscando mantener la misma funcionalidad que tenía anteriormente y a la vez ofrecer una experiencia más agradable mejorando la presentación. Para ello se ha hecho uso de las técnicas actuales de diseño web como uso de degradados.

Los cambios realizados han supuesto una mejora constatada tras dar a probar ambas interfaces a un grupo de personas. Este grupo se componía de 7 personas completamente ajenas al desarrollo de este proyecto y pudieron constatar la mejora del mismo.

Por otro lado, los cambios en los agentes destinados a mejorar el tiempo de respuesta del controlador han contribuido a que, en momentos de alta carga de trabajo, los tiempos de espera se reduzcan. Esto supone una mejor experiencia de usuario al eliminar posibles impaciencias por la tardanza en obtener una página del controlador. Para poder comprobar este último punto solo ha sido necesario enviar un hash al sistema para que se ponga en marcha y controlar el tiempo que tarda en devolver una página.

5.3. Mantenibilidad

La facilidad de mantener el sistema era un punto importante del proyecto por la falta de organización del mismo. Con los cambios que se han introducido se ha procurado mejorar este aspecto reduciendo el número de ficheros a cambiar para pequeños cambios.

Un ejemplo claro sería la introducción de un nuevo algoritmo, que antes suponía la modificación de al menos los siguientes ficheros:

- ControllerLink.cpp
- ComputeThreadProc.cpp

Además, puede ser necesario el cambio de más ficheros dependiendo de cómo se haya implementado el algoritmo.

Tras la reimplantación del sistema se simplifica enormemente el número de ficheros a modificar ya que ahora solo hace falta cambiar el del propio algoritmo. Igualmente, al añadir nuevas funcionalidades se simplifica enormemente ya que no hay necesidad de modificar línea alguna de código sobre el agente.

Igualmente el controlador se ha visto claramente beneficiado por el uso de CakePHP ya que, gracias al uso de una estructura ordenada en la organización de los elementos que componen el software, se puede ir directamente a la parte en la que pueda haber cualquier problema para solucionarla.

5.4. Algoritmo SHA-256

Las pruebas sobre este algoritmo han consistido, por una parte, en comprobar que los resúmenes que generaba eran correctos. Una vez confirmado este punto se ha procedido a calcular el tiempo que tarda en calcular un resumen para poder compararlo con el resto de algoritmos.

Para comprobar el tiempo que tarda se eligió una palabra de prueba con minúsculas, mayúsculas, números y algunos símbolos y una longitud de 8 caracteres. Tras esto se generó su resumen SHA-1 y SHA-256. Tras medir los tiempos que tardaba el sistema en encontrar la palabra para SHA-1 y SHA-256 se pudo determinar que la implementación de este último es 1,5 veces más lenta que la primera. Esto se debe principalmente a que el nuevo algoritmo requiere más pasos para obtener el resumen.

El resultado anterior es importante de cara a la implementación de nuevos algoritmos ya que será importante tener en cuenta cuanto más complejos son y, en caso de una complejidad excesiva, si es útil implementarlo. Si la complejidad de un algoritmo resultase muy elevada podría suceder que el tiempo que tardase en devolver un resultado fuera demasiado elevado. Esto significa que es importante comprobar con las ecuaciones mostradas en 2.1 para determinar de forma anticipada los tiempos.

Capítulo 6

Conclusiones

EL proyecto ha terminado existosamente tras haber hecho toda una serie de actividades que han permitido organizar y realizar el trabajo deseado. Estas actividades fueron planteadas cuando se inicio el proyecto y garantizan la buena organización del mismo.

A continuación puede verse un desglose ordenado de las actividades que han compuesto el proyecto:

- Búsqueda de soluciones existentes.

  El primer paso que se realizó fue buscar si existía alguna herramienta que, siendo software libre, permitiese hacer la evaluación de contraseñas utilizando tarjetas gráficas. Solo se encontró una, DHC.

- Estudio de la solución elegida.

  Disponiendo de una herramienta, se procedió a estudiar el código fuente de ésta para poder determinar como añadir nuevas funciones resumen y como extender las funcionalidades que éste ya tenía.

  Desgraciadamente, la documentación que tenía DHC abarcaba principalmente el proceso de compilación y ejecución, pero no detallaba cómo funcionaba por dentro por lo que se tuvo que hacer un gran esfuerzo en su estudio.

- Determinar las mejoras a realizar.

  Conociendo el código se podía empezar a proponer mejoras sobre el mismo. Las mejoras debían en todo momento garantizar que la herra-

mienta se adapte completamente a los requisitos propuestos (facilidad de ampliación, facilidad de mantenimiento, agilidad de uso, etc.).

Las mejoras propuestas para DHC han sido el núcleo de este proyecto y que ya se han visto anteriormente.

- Llevar a cabo las mejoras propuestas.

  A lo largo del capítulo 4 se ha podido ver el desarrollo de las mejoras propuestas, en qué han consistido y cómo se han llevado a cabo.

- Pruebas de aceptación.

  Para garantizar el buen funcionamiento de las modificaciones realizadas se ha tenido que comprobar que los resultados generados fueran los correctos

Tras la finalización del proyecto, y teniendo en cuenta los resultados, podemos concluir que todas las modificaciones aportadas han permitido mejorar sustancialmente la herramienta, pudiendo destacarse:

- La mejora en la mantenibilidad puede considerarse muy importante tras haberse simplificado de forma importante los puntos más críticos de la herramienta. Esto supone mayor facilidad para realizar cambios en caso de errores o de tener que introducir nuevas mejoras.

- Cuando se tiene un código que hace un uso intensivo de sentencias *if* anidadas, éste puede, en ciertos casos, ser sustituido por mecanismos de herencia. En el caso concreto del proyecto, esto ha permitido incorporar todo el sistema de algoritmos en sustitución de un mecanismo estático y complicado de modificar.

- El subsistema de *plugins* permite simplificar el código del agente, separando la funcionalidad de control del mismo de las tareas más específicas que se le soliciten. De este modo DHC se convierte en una herramienta más versátil a la que se le puede dotar de nuevas funcionalidades para las cuales no había sido diseñado inicialmente.

- El uso del CakePHP ha permitido simplificar el código del controlador gracias al uso de las herramientas que éste nos ofrece. Además, esto facilita enormemente el realizar cambios sobre la interfaz y mantener una mayor organización código.

- El uso de *frameworks* MVC facilita en gran medida el desarrollo de aplicaciones web al eliminar la necesidad de preocuparse por detalles de bajo nivel como puedan ser el acceso a base de datos.

- El nuevo diseño creado para DHC lo convierte en un sistema más atractivo, lo que anima a que sea utilizado. Este punto es importante ya que generalmente la gente se siente más a gusto con aquello que encuentra agradable frente a soluciones que puedan ser más completas.

- Tras el estudio y el uso de la herramienta se puede concluir que es muy importante la calidad de las contraseñas utilizadas ya que en su fortaleza depende una parte importante de la seguridad de muchas instituciones. Igualmente, el tipo de función resumen es también importante ya que las debilidades que ésta pueda tener puede afectar enormemente a la integridad de los datos.

- Las herramientas de control de versiones simplifican enormemente la gestión del código de los proyectos ya que eliminan la necesidad de mantener las versiones manualmente. Además, al funcionar este tipo de herramientas en red permite disponer del código en cualquier parte eliminando la necesidad de tener que recordar el llevar una copia encima. Por otra parte, también facilitan las tareas de vuelta atrás en el tiempo en caso de fallos graves, de este modo en caso de cometer un error se puede ir fácilmente a una versión anterior para deshacer los cambios.

Finalmente cabe concluir que tras el desarrollo de este proyecto se ha podido comprobar las dificultades que entraña el mantenimiento de herramientas con una escasa documentación.

Capítulo 7

Trabajos futuros

Tras terminar el proyecto hay muchas mejoras que han quedado pendientes o sería deseable poder añadir. A continuación puede verse una lista de aquellos elementos que se han considerado más importante de cara a la continuidad del proyecto:

Mejorar las comunicaciones entre agentes y controlador. Hasta este momento las comunicaciones entre agente y controlador hacen uso de un protocolo un poco pobre, en lo que a características se refiere, lo que supone también una restricción a la hora de hacer ampliaciones. La mejora propuesta busca ampliar este protocolo para que la variedad de funciones que pueda tener el agente sea mucho mayor.

Para desarrollar el nuevo protocolo habría que:

- Estudiar las mejoras que se quieren introducir para saber cómo afectarían a las comunicaciones (por ejemplo, para enviar *plugins* hay que poder indicarlo).
- Definir el nuevo protocolo (mensajes que envía el agente al controlador y las posibles respuestas a estos) con respecto al punto anterior.
- Realizar las modificaciones pertinentes en el controlador y en el agente. Para ello habrá que cambiar el fichero *agents_controller.php*, del controlador, y *ControllerLink.cpp*, del agente.

Carga de algoritmos desde el controlador. Esta extensión permitiría centralizar la administración de los algoritmos en el controlador de modo

que no haga falta tener que entrar de forma remota en cada uno de los agentes para añadir nuevas funcionalidades. Esto facilitaría la administración del sistema al evitar tener que acceder a cada agente para instalar las extensiones.

Para realizar esta tarea habría que modificar el controlador y el agente de modo que:

- El controlador debe saber en todo momento que algoritmos y *executors* tienen los agentes.
- El controlador pueda alojar los nuevos *plugins*.
- El controlador pueda enviar a los agentes los *plugins*.
- El agente debe poder cargar los *plugins* cuando se lo indique el controlador.

Para esto habría que cambiar el protocolo de comunicación que se utiliza actualmente.

Crear nuevos algoritmos de comprobación de hashes. En la actualidad hay una gran cantidad de algoritmos de resumen que pueden ser portados a DHC y sería interesante disponer de ellos.

Estos algoritmos se pueden crear fácilmente dentro de *plugins* que serían cargados posteriormente. Para ello cada nuevo algoritmo deberá heredar de la clase *Algorithm*, implementar su funcionalidad y ser exportado como *plugin*.

Crear nuevos mecanismos de ejecución. En estos momentos los algoritmos solo pueden hacer uso de un sistema básico de ejecución, pero podrían implementarse nuevos sistemas que permitiesen mejoras en los tiempos o simplemente implementar nuevos algoritmos más allá de las funciones resumen.

Estos nuevos mecanismos de ejecución pueden implementarse de distintos modos dependiendo del objetivo. Por ejemplo, en el caso de querer realizar optimizaciones sobre el sistema existente actualmente habría que:

- Estudiar el *Executor* que se quiere mejorar para buscar sus cuellos de botella, variables innecesarias, etc.

- Crear un nuevo *Executor* con las modificaciones que implementen las mejoras.
- Probar los cambios realizados.

En caso de querer crear un nuevo *Executor* para una funcionalidad aún no implementada, los pasos serán los mismos, pero hay que estudiar como deberá ejecutarse la nueva funcionalidad.

Implementar nuevos protocolos de seguridad. DHC no tiene porque restringirse sólo a funciones resumen. Con esto se conseguiría que la herramienta abarcase un mayor número de mecanismos de seguridad como podrían ser:

- Contraseñas de redes WiFi con protección WPA.
- Determinar las claves generadas a partir de un *handshake* del sistema TLS utilizado en páginas web.

Para este propósito habrá que hacer uso de los puntos anteriores ya que puede que se deba cambiar el protocolo de comunicaciones además de añadir nuevos algoritmos y *executors*.

Control de cambios sobre el sistema de ficheros. Esta característica puede utilizarse para determinar cuando un *plugin* ha cambiado y volverlo a cargar sin tener que reiniciar el agente. De este modo el rendimiento de la aplicación mejoraría al no tener que parar casi nunca.

El desarrollo de esta mejora podría hacerse del siguiente modo:

- Se debe determinar las llamadas al sistema que permiten monitorizar los cambios sobre el sistema de ficheros.
- Hay que diseñar un mecanismo que permita eliminar un *plugin* de la memoria para ser sustituido por otro. Hay que tener en cuenta que esto supone saber si está o no en uso y que cuando se realice el cambio no podrán ser utilizados.
- Hay que desarrollar y probar la solución propuesta.

Sistema de controladores jerárquicos. El objetivo de este cambio es poder disponer de un árbol de controladores donde los que se encuentren

en las ramas sean los encargados de repartir el trabajo entre los agentes y los demás controladores simplemente repartan trabajo entre ellos. El objetivo es repartir la carga en casos en los que haya una gran cantidad de agentes.

Este sistema necesita que se cambien los controladores, de tal modo que puedan comunicarse también entre ellos. Hay varias opciones para conseguir esto:

- Crear dicha funcionalidad y ejecutarla desde un planificador de tareas.
- Esperar un número de peticiones de los agentes para realizar la llamada al controlador superior. Esta llamada se realizaría durante la solicitud del agente ya que PHP no puede crear tareas en paralelo para este tipo de acciones.

La elección de una u otra solución dependerá del uso que se vaya a dar al sistema, por lo que habrá que estudiar como afectaría al rendimiento cada caso.

Apéndice A

Presupuesto

A continuación se va a presentar el presupuesto requerido para llevar a cabo el proyecto.

La duración total del proyecto ha sido de 11 meses, teniendo en cuenta que la dedicación que se ha aplicado ha sido de aproximadamente 13,2 horas semanales (un tercio de las 40 horas semanales).

A.1. Desglose de actividades del proyecto

Para calcular el total de horas del proyecto se ha tenido en cuenta todas las actividades realizadas en el mismo. En el cuadro A.1 puede apreciarse el trabajo realizado, que ha ido desde el estudio de la solución previa, el análisis de los posibles cambios a realizar, el diseño de las modificaciones realizadas hasta la instalación del sistema de pruebas (la máquina Tesla).

Actividad	Horas
Estudio DHC	80 h
Anásis y diseño	70 h
Implementación	280 h
Pruebas	40 h
Instalación sistema	5 h
Documentación	100 h

Cuadro A.1: *Desglose de horas por actividad*

Cargo	Horas	Coste/Hora	Total
Analista	150 h	16,00€/hora	2.400,00€
Programador	280 h	16,00€/hora	4.480,00€
Responsable documentación	100h h	16,00€/hora	1.600,00€
Responsable pruebas	40 h	16,00€/hora	640,00€
Técnico instalación	5 h	16,00€/hora	80,00€
Total	575 h		9.200,00€

Cuadro A.2: *Desglose de horas por actividad*

Servicio	Precio mes	Meses	Total
Servidor privado virtual	16,99€/mes	11	186,89€
Total			186,89€

Cuadro A.3: *Coste de los servicios subcontratados*

A.2. Gasto en personal imputable al proyecto

Para ajustar el precio por hora del programador se ha hecho una búsqueda entre las ofertas de trabajo que se encuentran actualmente en internet. Se ha tomado como muestra aquellas en las que se busca profesionales con experiencia donde el sueldo ronda los 36.000€/año.

El coste total por personal es de 9.200€ como puede apreciarse en el cuadro A.2.

A.3. Servicios subcontratados

En el cuadro A.3 puede apreciarse el coste de los servicios que se han subcontratado.

A.4. Recursos materiales empleados

Durante la realización de este proyecto se ha hecho uso de una gran cantidad de software. Al haber sido software libre y gratuito el coste repercutido a un posible cliente es de 0€ y no hace falta amortizarlo lo que puede ser considerado una ventaja importante frente a otros sistemas que pueden encarecer significativamente el producto.

Recurso	Cantidad	Coste total
Tesla T1070	1	7.082,00€
Servidor Tesla	1	2.528,00€
Total		9.610,00€

Cuadro A.4: *Coste de los recursos materiales empleados*

Recurso	Precio	Amortización (meses)	Uso	Repercutido
MacBook Pro	2.600,00€	60	60 %	42,05€
Sobremesa	1.200,00€	60	40 %	28,75€
Total				70,80€

Cuadro A.5: *Coste por amortización de equipos*

Por lo anterior, solo es necesario repercutir el coste de los dispositivos materiales empleados, como puede verse en el cuadro A.4.

A.5. Amortizaciones

La amortización es la perdida de valor del equipo por su uso y su antigüedad. Por este motivo debe ser repercutida en el proyecto y que así no se pierda dinero por la compra de los mismos.

Se ha considerado que un mes tiene en torno a las 160 horas laborables (4 semanas de 40 horas cada una) para la realización del cuadro A.5.

A.6. Gastos indirectos

Se debe tener en cuenta el coste de todos aquellos elementos que, si bien no repercuten directamente sobre el proyecto, suponen un gasto constante durante la vida de éste.

El precio de la conexión a internet y de la telefonía se ha calculado a partir de las facturas obtenidas de unos 40€/mes cada una. Considerando que un mes tiene 4 semanas a 40 horas laborables el precio por hora es de 0,25€/hora. Este valor se multiplica por el total de horas empleadas para realizar el proyecto.

El material de oficina comprende folios, cuadernos e instrumental para escribir. Apenas se ha gastado material de este tipo (un cuaderno, algunos

Descripción	Coste
Conexión a internet	143,50€
Llamadas telefónicas	143,50€
Material de oficina	10,00€
Alquiler local	3.953,13€
Electricidad	74,75€
Servicio de limpieza	3.593,75€
Total	7.918,63€

Cuadro A.6: *Costes indirectos*

Descripción	Coste
Personal	9.200,00€
Subcontratas	186,89€
Amortizaciones	70,80€
Material	9.610.00€
Indirectos	7.918,63€
Total	26.986,32€

Cuadro A.7: *Resumen de los gastos del proyecto*

folios y un bolígrafo) por lo que su valor es bajo.

El alquiler de locales en Leganés están sobre los 1.100€/mes de media (obtenido a partir de una revisión de `www.idealista.com`). Estos locales sólo pueden amortizarse durante la actividad económica del mismo (40h/semana). Esto supone un coste por hora de 6,875€/hora.

Se ha supuesto un gasto en electricidad de unos 20€/mes. Esto supone un coste de 0,13€/hora.

La limpieza del local puede rondar en torno a los 1.000€/mes. Se ha considerado que la subcontratación del servicio de limpieza puede rondar los 25€/hora y no sería necesario más de 2 horas al día durante cada día de la semana (unos 20 días al mes). Esto supone un coste de 6,25€/hora.

En el cuadro A.6 hay un resumen de los gastos indirectos del proyecto.

A.7. Resumen del presupuesto

Con todos los datos juntos (ver cuadro A.7) procedemos a calcular el precio de venta de la solución desarrollada.

Lo primero de todo es que pueden darse imprevistos a lo largo del desarrollo, como roturas de material, enfermedad del personal, etc. Por este motivo se aplica un margen del 10 % sobre el precio para compensar estos posibles imprevistos: 2.698,63€.

$$\text{Coste Total} + \text{Margen de imprevistos} = 29.684{,}95€$$

El margen de beneficio a aplicar es del 25 % sobre el precio con imprevistos: 7.421,24€.

$$\text{Precio Final} = \text{Precio con Imprevistos} + \text{Beneficio} = 37.106{,}19€$$

El precio final de la solución, contando el 18 % de I.V.A., es: **43.785,30€**.

Apéndice B

Entorno de desarrollo

Para el desarrollo del proyecto se ha hecho uso de una gran cantidad de herramientas y tecnologías que es importante ver para poder comprender mejor el modo de desarrollo.

B.1. CMake

CMake es una herramienta para la construcción de ficheros Makefile. El mayor inconveniente que hay a la hora de utilizar ficheros Makefile es que resulta complicado mantenerlos y aún más crear versiones para diferentes sistemas operativos. Este último punto se debe a que pueden variar sutilmente algunas de las reglas para crear el fichero y que dependiendo del sistema operativo se puedan necesitar unas u otras librerías. Además, utilizando ficheros Makefile es complicado realizar comprobaciones sobre el sistema, como pueden ser comprobar si está o no instalada una biblioteca en concreto.

Existen varias soluciones para la generación de ficheros Makefile, entre ellas destaca GNU Autotools. Ésta es probablemente una de las herramientas más famosas en el mundo del software libre por ser la que se utiliza dentro del proyecto GNU. Desgraciadamente Autotools es una herramienta complicada de utilizar lo que nos hizo decantarnos por CMake.

CMake utiliza una sintaxis sencilla para definir las reglas de comprobación, asignación de dependencias y cualquier cosa que tenga que ver con el proceso de generación del proyecto.

Para instalar CMake en el sistema se ejecutará el siguiente comando:

```
$ sudo apt-get install cmake
```

B.2. Git

Una parte muy importante de todo proyecto es la gestión de los ficheros que se están utilizando. En el caso de un proyecto software el control de los cambios es fundamental para evitar problemas.

Git es un sistema de control de versiones desarrollado por Linus Torvalds para el núcleo Linux. Sus principales características son:

- Es un sistema distribuido, lo que evita la necesidad de disponer siempre de un servidor central al que enviar los cambios, como sucede con otros sistemas como CVS o Subversion.
- Facilita la creación de ramas (*branch*) de desarrollo y la fusión de éstas (*merge*) cuando es necesario de forma rápida y sencilla.
- Facilita disponer de copias de seguridad distribuyéndolas entre varios equipos, al menos en el uso que se ha dado para este proyecto final de carrera.

Para poder utilizar git solo hace falta instalarlo ejecutando el siguiente comando:

```
$ sudo apt-get install git-core
```

Durante el desarrollo del proyecto se han utilizado varias ramas para controlar las distintas partes. Concretamente se ha utilizado:

master Mantiene el código que se ha comprobado que funciona correctamente. No se puede incorporar nada a esta rama directamente, sino que hay que desarrollar primero en otra rama, comprobar correcto funcionamiento de los cambios y luego se realiza la mezcla de los cambios con *master*s.

ldopen En esta rama se realizaron los cambios para soportar *plugins*. La nomenclatura de ldopen proviene de la llamada a sistema utilizada para poder desarrollar esta funcionalidad.

nuevaweb Aquí se encuentra todo el código de pruebas del nuevo controlador desarrollado así como cambios sobre los agentes para soportar la nueva arquitectura.

fixcompiling Cambios realizados para que la compilación en el sistema operativo Mac OS X funcionara correctamente.

B.3. Apache

Para poder hacer uso del controlador es necesario disponer de un servidor web con soporte de PHP. En este caso se ha utilizado el servidor web Apache en su versión 2 por ser el más popular, disponer de una abundante documentación y ser software libre.

Cualquier distribución de GNU/Linux puede instalar este servidor web con unos pocos *clicks* de ratón o en su defecto un único comando. En el caso de las distribuciones basadas en Debian solo hace falta ejecutar el siguiente comando:

```
$ sudo apt-get install apache2
```

B.4. Redmine

Redmine es un gestor de proyectos realizado en Ruby on Rails. Permite la asignación de tareas, calendarios, documentación y otras muchas funcionalidades más a través de plugins. Como la mayoría de las herramientas utilizadas es software libre.

Esta herramienta ha sido de utilidad para poder controlar los problemas que han ido surgiendo y que no quedarán pendientes.

La instalación de Redmine es probablemente de las más complicadas de todas las herramientas utilizadas en este proyecto. Para instalar esta herramienta se ha tenido que seguir los siguientes pasos:

- Instalación de ruby, rails y rake:

  ```
  $ sudo apt-get install ruby rails rake
  ```

- Instalación del módulo *passenger* para apache[1]:

  ```
  $ sudo gem install passenger
  $ sudo passenger-install-apache2-module
  ```

[1]Passenger dispone de paquetes para algunas distribuciones. Por desgracia, la distribución que utilizaba el servidor de Redmine no disponía de dichos paquetes

B.5. CUDA

Como no, se ha hecho uso de las herramientas de NVIDIA para compilar el proyecto. El *framework* CUDA trae las bibliotecas básicas necesarias para poder comunicarse con la tarjeta gráfica y las herramientas de compilación.

Para poder utilizar CUDA hay que instalar previamente los controladores de la tarjeta gráfica adecuados ya que los controladores que vienen por defecto solo permiten funciones gráficas.

Para desarrollar el proyecto se ha hecho uso de la versión 2.3 de CUDA que puede encontrarse en `http://developer.nvidia.com/object/cuda_3_2_downloads.html`. Desde esta página se pueden descargar tanto los controladores para el sistema operativo como las herramientas necesarias para desarrollar en CUDA.

Es importante comprobar antes de utilizar CUDA si la tarjeta gráfica que tiene el equipo es compatible.

Para instalar CUDA en linux se necesita instalar primero el controlador de la tarjeta gráfica. Este controlador requiere las cabeceras del núcleo Linux para poder compilarse. Por este motivo hay que asegurarse de que ya se encuentren instaladas y en caso contrario se pueden instalar con el siguiente comando:

```
$ sudo apt-get install linux-headers-generic
```

Por desgracia no se ofrecen paquetes adaptados a cada distribución por lo que hay que utilizar el instalador que ofrece NVIDIA:

```
$ wget http://developer.download.nvidia.com/compute/cuda/
3_2_prod/drivers/devdriver_3.2_linux_32_260.19.26.run
$ sudo sh ./devdriver_3.2_linux_32_260.19.26.run
```

Tras este paso solo hay que seguir las indicaciones del instalador.

Hay ocasiones en las que el instalador de los controladores de NVIDIA fallan y no crean los dispositivos necesarios en */dev*, por lo que hay que crearlos a mano. Para ello se puede utilizar las siguiente llamadas en línea de comando:

```
$ for i in $(seq 0 9); do sudo mknod /dev/nvidia${i} c 195 ${i}; done
$ sudo mknod /dev/nvidiactl c 195 255
```

Una vez instalado el controlador de CUDA se puede proceder a instalar las herramientas. Éstas utilizan su propio instalador, pero está adaptado a distintas distribuciones, por lo que hay que elegir el más adecuado:

```
$ wget http://developer.download.nvidia.com/compute/cuda/
3_2_prod/toolkit/cudatoolkit_3.2.16_linux_64_ubuntu10.04.run
$ sudo sh ./cudatoolkit_3.2.16_linux_64_ubuntu10.04.run
```

Es recomendable instalar las herramientas en los directorios que nos indique como predeterminados (generalmente */usr/local/cuda*).

B.6. Compilador de C y C++ de GNU

El compilador de GNU es probablemente uno de los más utilizados hoy día en el mundo del software libre. Este compilador dispone de herramientas para compilar una gran cantidad de lenguajes distintos como C, C++, Objective-C, Pascal y muchos más. Es el compilador que traen todas las distribuciones GNU/Linux en la actualidad y hasta sistemas operativos como Mac OS X lo utilizan.

La mayor parte del código empleado en el proyecto está escrito en C++ por lo que se ha necesitado hacer uso del compilador de C++ de GNU, principalmente por ser el más sencillo de instalar en Linux.

Para asegurarnos que al instalar el compilador de GNU se instalen todas las dependencias que necesitamos podemos ejecutar la siguiente orden:

```
$ sudo apt-get install build-essential g++
```

El paquete *build-essential* se encarga de instalar todas las herramientas y bibliotecas de compilación básicas. Luego le indicamos que se asegure de instalar g++ (nombre que tiene el compilador de C++ suministrado por GNU).

B.7. NASM

NASM es un ensamblador multiplataforma que se necesita para poder generar la versión de CPU de MD5. Es importante asegurarse de utilizar la última versión de la herramienta ya que, por ejemplo, la versión que trae Mac OS X no está actualizada y no permite realizar compilaciones de 64 bits.

Para instalarlo solo hay que ejecutar:

```
$ sudo apt-get install nasm
```

B.8. PHP

PHP es un lenguaje de programación web muy extendido gracias a que es gratuito y libre. Tiene una sintaxis sencilla fuertemente basada en C y es muy fácil de aprender.

Es muy habitual hablar de instalaciones LAMP, que son aquellas que hacen uso de Linux, Apache, MySQL y PHP y que son ampliamente utilizadas en internet por gran número de servicios de alojamiento por su bajo coste y facilidad de instalación y configuración.

Al ser un lenguaje muy extendido se ha utilizado para la creación del nuevo controlador de DHC. El anterior controlador también hacía uso de PHP pero cuando se planteó crear uno nuevo se barajaron soluciones basadas en Ruby y Groovy. Estas soluciones fueron descartadas por ser algo más complicadas de configurar. Para ser más precisos, Ruby requiere que se instalen módulos especiales para Apache y hay que revisar muy bien la configuración. En el caso de Groovy se necesita un servidor Apache Tomcat basado en java que puede suponer una gran sobrecarga para el sistema.

Para instalar PHP e integrar éste con Apache solo hace falta ejecutar el siguiente comando en cualquier distribución basada en Debian:

```
$ sudo apt-get install php5
```

B.9. CakePHP

CakePHP es un *framework* de desarrollo web que hace uso del lenguaje de programación PHP y el patrón arquitectónico MVC. Con CakePHP se pueden hacer aplicaciones fácilmente sin tener que preocuparse de cosas como el acceso a base de datos o el control de flujo a través de la aplicación.

Gracias al uso de CakePHP se ha podido centrar la atención del controlador más en las tareas propias del mismo. De este modo ahora no hay que preocuparse de como hacer el acceso a la base de datos o incluso el tipo de base de datos que se está utilizando.

Para utilizar CakePHP solo hace falta descargarlo y descromprimirlo en un directorio. Éste ya dispone de una estructura básica de directorios para

utilizar directamente en un proyecto. Para este proyecto los directorios más importantes son:

app/controllers Almacena los controladores de la aplicación. Estos controladores se encargan de la lógica de negocio.

app/views Representan las salidas que producirá la aplicación.

app/models Describe la estructura de la base de datos (las tablas y sus relaciones).

app/config Tiene la configuración del motor de base de datos que hay que utilizar y la configuración de la misma. Por otra parte se ha almacenado aquí el script para generar las tablas en esta base de datos.

B.10. MySQL

MySQL es uno de los motores de base de datos libres más extendidos de internet. La principal razón de esto es que es fácil de administrar y que históricamente siempre ha sido una base de datos muy rápida, lo que suponía una ventaja para las aplicaciones web con una alta carga de peticiones.

El controlador de DHC hace uso de bases de datos para almacenar la información de las tareas que se están ejecutando y las estadísticas de las mismas.

Su instalación es tan sencilla como ejecutar:

```
$ sudo apt-get install mysql-server
```

B.11. Gimp

Gimp es una herramienta de retoque fotográfico de una gran calidad y que desde hace muchos años se la considera el Photoshop del software libre.

Para las imágenes utilizadas en la web se ha utilizado Gimp por ser una herramienta potente y relativamente sencilla de utilizar.

Para su instalación se ha utilizado:

```
$ sudo apt-get intall gimp
```

B.12. LaTeX

Es una de las herramientas más extendidas para la creación de textos científicos. LaTeX son un conjunto de reglas para la herramienta TeX creada por Donald E. Knuth. Su diseño está pensado centrarse en el contenido de los documentos en lugar de en su diseño. Es muy parecido a lo que ahora se hace en web con HTML+CSS.

La memoria de este proyecto final de carrera ha sido escrito utilizando LaTeX por las ventajas que ofrece frente a otros sistemas como Microsoft Word o LibreOffice. Entre estas ventajas podemos destacar que al almacenar solo ficheros de texto es más sencillo realizar un control de versiones sobre las modificaciones, lo que resulta más complicado utilizando ficheros binarios.

Para instalar el compilador de LaTeX en el sistema hay que ejecutar:

```
$ sudo apt-get install texlive-latex-extra
```

B.13. GnuPlot

GnuPlot es una herramienta para la generación de gráficas de funciones. Ha sido utilizada para la gráfica de la distribución de la paradoja del cumpleaños.

Esta herramienta es muy útil cuando se necesita hacer gráficas de datos y no se tiene un sistema muy potente ya que requiere de poca memoria. Un ejemplo de ello es cuando se tienen grandes tablas de datos en MS Office lo que hace que tarde en generar gráficas o incluso manejar los datos.

En este caso se ha creado un script de GnuPlot que ha sido integrado dentro del sistema de compilación automático de CMake para generar un PDF y poder ser incrustado en el documento LaTeX.

Para instalar GnuPlot sólo hace falta utilizar la siguiente orden en el sistema:

```
$ sudo apt-get install gnuplot
```

B.14. LibreOffice

LibreOffice es una herramienta que ha nacido a partir del código de OpenOffice.org. Su creación se motivo en el descontento de los desarrolladores

por la falta de una estrategia por parte de Oracle tras la compra de Sun (y con ello de OpenOffice.org). Aparentemente, Oracle no tiene mucho interés en apoyar el software libre adquirido por la compra de Sun Microsystems.

Todas los diagramas que aparecen en la memoria ha sido realizadas con la aplicación de dibujo de LibreOffice.

LibreOffice puede ser descargado desde su página web en `http://www.libreoffice.org/download/`.

B.15. UML

UML, siglas de *Unified Modeling Language*, es un lenguaje gráfico para modelar software ampliamente utilizando en el mundo de la programación orientada a objetos.

Algunos de los diagramas expuestos en la memoria hacen uso de dicho lenguaje. Concretamente se ha hecho uso de diagramas de clases y de secuencia. El primero permite expresar de forma gráfica los elementos que forman todo, o una parte, de un programa, pero sin entrar al detalle de como deben comportarse. Los diagramas de secuencia representan un caso de funcionamiento concreto y son útiles para poder mostrar cómo actúa una parte del proyecto.

B.16. Debian y Ubuntu

Debian es una de las distribuciones de Linux más antiguas y su filosofía es utilizar sólo software libre. Uno de los puntos más importantes a destacar sobre esta distribución es que está mantenida enteramente por voluntarios y que tiene uno de los mayores repositorios de aplicaciones de todas las distribuciones existentes.

Debian posee muchas distribuciones derivadas que sobre ésta añaden paquetes y/o hacen cambios de configuración. Una de estas distribuciones, y probablemente la distribución más famosa actualmente, es Ubuntu.

Para descargar Debian se puede ir a `http://www.debian.org/distrib/` y Ubuntu se puede obtener de `http://www.ubuntu.com/desktop/get-ubuntu/download`.

B.17. Organización del entorno de trabajo

Para realizar el proyecto se ha utilizado, a parte de las herramientas anteriormente mencionadas, una serie de equipos organizados del siguiente modo:

Servidor web Se dispone de un servidor privado virtual que tiene instalado Apache, Git, Redmine, PHP y la versión de pruebas del controlador de DHC.

Este servidor ha sido subcontratado para garantizar un buen funcionamiento del servicio ya que la red de la universidad no ofrece garantías suficientes de estabilidad. Además, el uso de un servidor privado virtual ofrece la posibilidad de instalar todas aquellas herramientas que sean necesarias.

Equipos de desarrollo Puede ser cualquier equipo desde el que puede escribirse código fuente. Se han utilizado dos ordenadores para este propósito uno con Mac OS X y el otro con Kubuntu 10.10.

Desde estos equipos se realizan constantes peticiones de *pull* y *push* a git para mantener siempre una copia actualizada de los cambios.

Equipo de ejecución Este equipo se encuentra instalado en el laboratorio de Evalues en el Parque Cientifico-Tecnológico de Leganés. Dispone de 2 microprocesadores AMD Opteron de 4 núcleos cada uno, 8 GB de RAM y una NVIDIA Tesla S1070.

Tanto la unidad Tesla como el equipo que la controla pueden montarse en un *rack* de ordenadores.

Para ejecutar las pruebas este ordenador hace peticiones de *pull* al git para obtener las versiones más actuales del código.

Para comprender mejor como interactúan entre sí los distintos se puede consultar la figura B.1.

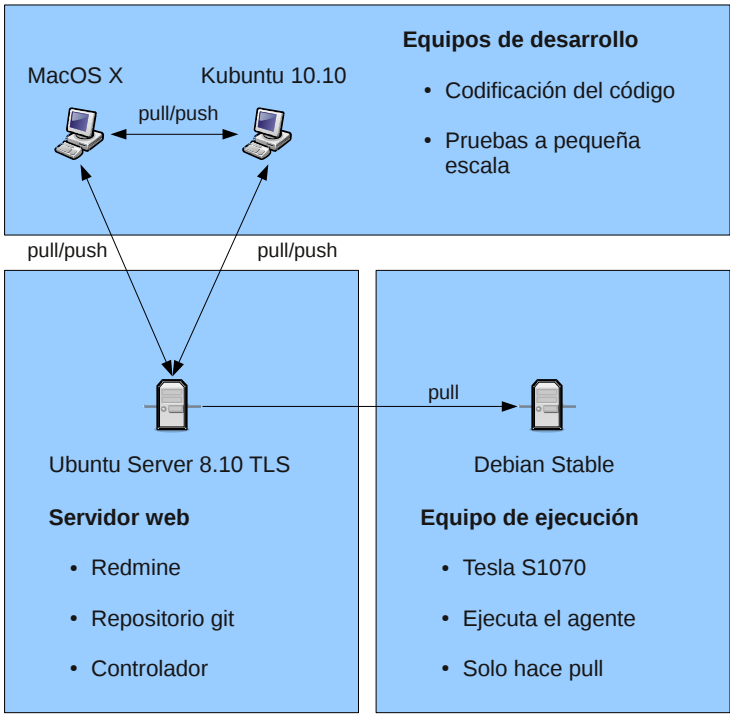

Figura B.1: *Diagrama del entorno de trabajo*

Apéndice C

Manual de usuario

En este capítulo se va a mostrar como instalar y usar DHC. Además se verán algunos detalles más avanzados de configuración para casos especiales.

C.1. Instalación

Antes de poder hacer uso de la aplicación es necesario instalarla. En las siguientes subsecciones se explicarán los pasos detallados para poner a punto el sistema.

C.1.1. Instalación del agente

El agente es uno de los elementos esenciales de DHC y el que entraña mayor dificultad de instalación. Por ese motivo hay que prestar especial detalle al procedimiento para identificar posibles fallos.

El proceso de instalación consiste en la instalación del sistema operativo en la máquina, los controladores del mismo y las herramientas de compilación tal y como se ha descrito en el apéndice B.

Una vez instaladas todas las herramientas necesarias se procede a la compilación del agente. Para este propósito se deberán seguir los siguiente pasos:

1. Se hace una copia del repositorio para trabajar con ella:

```
$ git clone $LOCATION dhc
```

$LOCATION hace referencia al lugar en el que se encuentre el repositorio y puede ser de la forma *file://directorio* o *usuario@ssh://direccion/ruta*. Una vez realizada esta copia dispondremos de una versión local del respositorio sobre la que trabajar. Esta versión se habrá creado en el directorio en el que estuviéramos en el momento de realizar la llamada.

2. Preparamos la compilación:

```
$ cd dhc/v3
$ cmake .
$ mkdir ptx
```

3. Compilamos el código:

```
$ make
```

4. Una vez compilado el código disponemos en el directorio del proyecto de una carpeta llamada *bin* y otra *ptx* que contienen los binarios del sistema operativo y de CUDA. Para terminar la instalación procederemos del siguiente modo:

```
$ sudo cp bin/agent /usr/bin
$ sudo mkdir -p /usr/lib/cracker/ptx
$ sudo cp bin/*.aplug.so /usr/lib/cracker/ptx
$ sudo cp ptx/* /usr/lib/cracker/ptx
```

En el cuadro C.1 podemos ver la lista de tareas a realizar. Este cuadro puede ser utilizado como *checklist* de la instalación.

C.1.2. Instalación de controlador

El controlador es la parte encargada de la gestión del sistema. Es algo más sencilla de instalar que el agente al no requerir ninguna compilación, pero en cambio necesita que se modifiquen ciertos ficheros.

Para poder utilizar el controlador hace falta que el ordenador disponga del servidor web Apache y PHP. Además puede tener MySQL o se puede alojar la base de datos en otro ordenador. Con independencia de dónde se decida instalar MySQL se considerarán las siguientes variables que utilizaremos para identificar datos de configuración:

Hecho	Descripción	Notas
	Instalar S.O. Linux en el ordenador	
	Instalar las herramientas de desarrollo de g++ y cmake	
	Comprobar e instalar, si es necesario, las cabeceras de Linux	
	Instalar controladores de NVIDIA con soporte de CUDA	
	Comprobar si se han creado los dispositivos de NVIDIA en /dev	
	Compilación del agente	
	Instalación de los binarios	

Cuadro C.1: *Matriz de comprobación de la instalación del agente*

PWEB Ruta al punto en que se encontrará el controlador y que es un directorio accesible desde internet (normalmente es /var/www).

WEB_HOST Dirección del servidor web para la base de datos (si el servidor aloja ambos servicios se considerará que es localhost).

MYSQL_HOST Dirección IP de la máquina que aloja el servidor de base de datos (en caso de ser el mismo equipo que el servidor web se considerará que es localhost).

DB_USER Usuario que se configurará en MySQL para poder acceder a la base de datos del controlador.

DB_PASS Contraseña del usuario $DB_USER.

DB_NAME Nombre que tendrá la base de datos (por ejemplo, DHC).

Los pasos a seguir para la instalación son los siguientes:

1. Instalar el sistema operativo en el servidor de Apache y de MySQL según lo mostrado en el apéndice B.

2. Creamos la base de datos en el servidor de MySQL:

```
$ mysql -h $MYSQL_HOST -u root -p
mysql> create database $DB_NAME
mysql> grant all privileges on $DB_NAME.* to ‘$DB_USER‘@‘$WEB_HOST‘ ident
mysql> quit
```

3. Se hace una copia del repositorio para trabajar con ella:

   ```
   $ git clone $LOCATION dhc
   ```

   $LOCATION hace referencia al lugar en el que se encuentre el repositorio y puede ser de la forma *file://directorio* o *usuario@ssh://direccion/ruta*. Una vez realizada esta copia dispondremos de una versión local del respositorio sobre la que trabajar. Esta versión se habrá creado en el directorio en el que estuviéramos en el momento de realizar la llamada.

4. Copiamos el controlador a su destino:

   ```
   $ sudo cp -r dhc/v3/controller/* $PWEB
   ```

5. Volcamos la base de datos inicial:

   ```
   $ mysql -h $MYSQL_HOST -u $DB_USER \
     -p $DB_PASS < $PWEB/app/config/database.sql
   ```

6. Editamos el fichero de configuración de la base de datos ($PWEB/app/config/database.php) para que quede como sigue:

   ```
   class DATABASE_CONFIG {
     var $default = array(
       'driver' => 'mysql',
       'persistent' => false,
       'host' => '$MYSQL_HOST',
       'login' => '$DB_USER',
       'password' => '$DB_PASS',
       'database' => '$DB_NAME',
       'prefix' => '',
     );
   ```

Hecho	Descripción	Notas
	Instalar S.O. Linux en el servidor web	
	Instalar S.O. Linux en el servidor de MySQL	
	Configurar MySQL	
	Copiamos e instalamos el controlador	
	Iniciamos la base de datos	
	Configuramos el acceso a la base de datos	

Cuadro C.2: *Matriz de comprobación de la instalación del controlador*

```
    var $test = array(
      'driver' => 'mysql',
      'persistent' => false,
      'host' => '$MYSQL_HOST',
      'login' => '$DB_USER',
      'password' => '$DB_PASS',
      'database' => '$DB_NAME',
      'prefix' => '',
    );
  }
```

A partir de este momento el controlador ya se encuentra listo para ser utilizado. En el cuadro C.2 podemos ver un resumen de los pasos a seguir.

C.2. Uso de DHC

Para utilizar DHC se hace uso del controlador. Éste es la interfaz que permitirá utilizar todas las funcionalidades que nos ofrezcan los agentes. Pero antes de esto hay que iniciar los agentes.

La inicialización de un agente es tan sencilla como ejecutar el siguiente comando en el ordenador en que se encuentre:

```
$ agent $DIRECCION_CONTROLADOR
```

Figura C.1: *Pantalla principal de DHC*

Donde $DIRECCION_CONTROLADOR es la URL del controlador.

Una vez que se tiene a los agentes en funcionamiento se puede proceder a utilizar el controlador. Para este fin se hará uso de un navegador web[1]

Nada más entrar en la aplicación podemos observar una pantalla como la mostrada en la figura C.1. En caso de que el sistema esté en funcionamiento podremos ver a qué velocidad está trabajando (en millones de hashes por segundo) o nada en otro caso.

El primer paso es ir a enviar para solicitar la realización de alguna tarea (ver figura C.2). En este punto tendremos que seleccionar todas las opciones de configuración de la tarea.

Tras el envío de la tarea el sistema nos dirá si ésta se ha almacenado

[1] Actualmente existe una gran cantidad de navegadores web en el mercado. Se recomienda encarecidamente el uso de navegadores con soporte de HTML5 y CSS3 como pueden ser Google Chrome, Apple Safari o Mozilla Firefox.

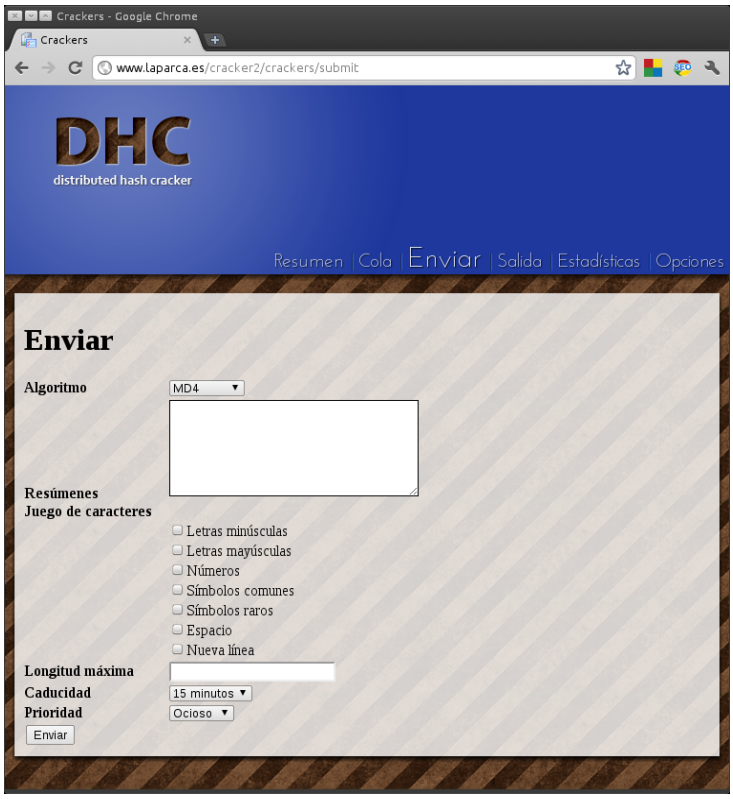

Figura C.2: *Envío de una tarea*

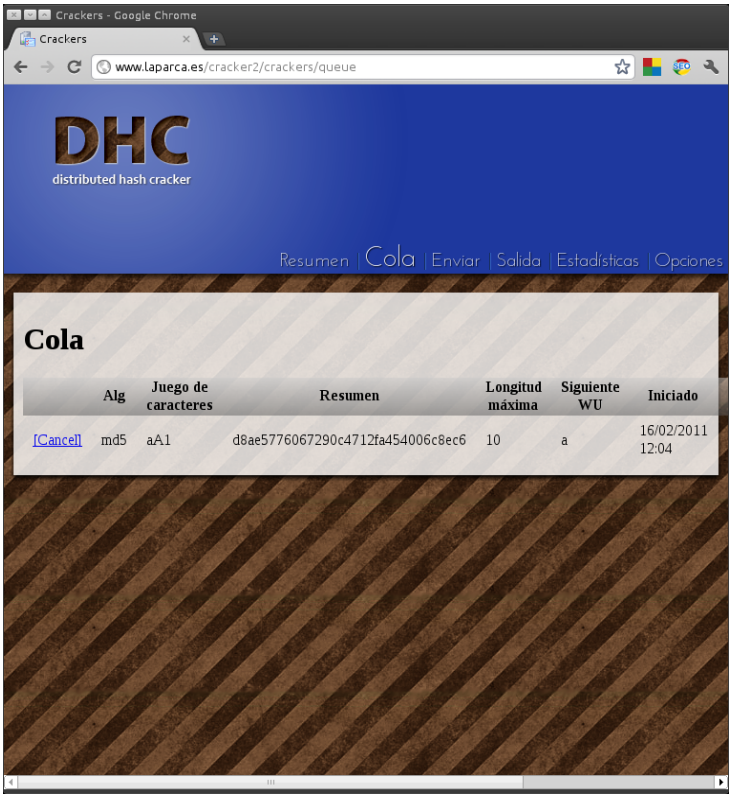

Figura C.3: *Cola de tareas*

correctamente o si a habido algún error.

En todo momento podemos comprobar qué tareas hay en funcionamiento desde la pantalla de cola (figura C.3).

Finalmente, podemos comprobar las tareas que se han terminado en la sección salida (ver figura C.4). Además, se puede comprobar la razón de la finalización de la tarea (resultado encontrado, no encontrado, etc.).

Si en algún momento queremos conocer mejor la situación de los agentes, se puede comprobar en estadísticas (ver figura C.5) el rendimiento de cada uno y el último momento en que se ha sabido de ellos.

C.3. Administración de *plugins*

Durante el tiempo que esté en uso DHC pueden surgir ocasiones en las que se necesite instalar nuevos *plugins* o eliminar los antiguos. Para ello se

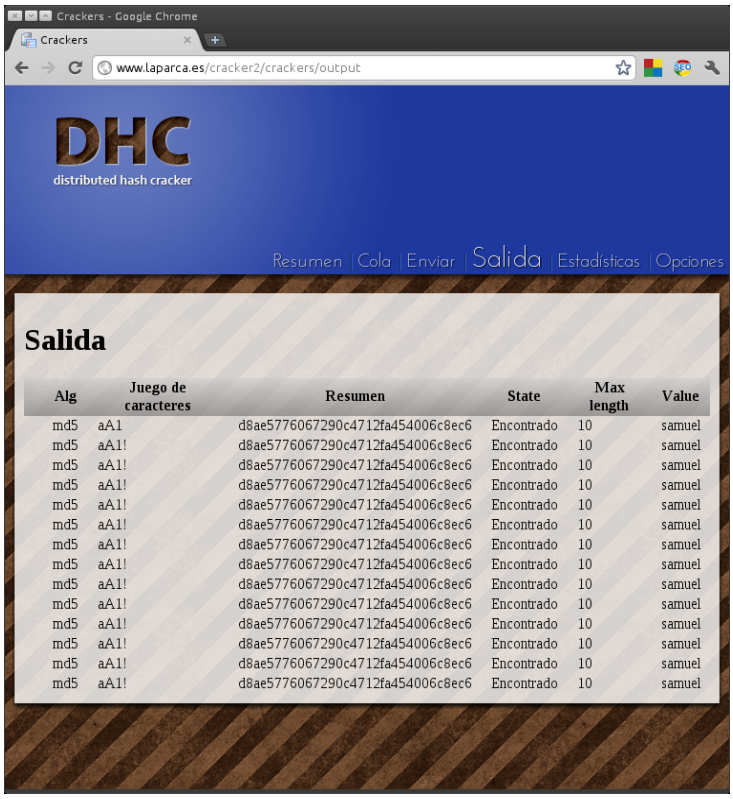

Figura C.4: *Salida de las tareas*

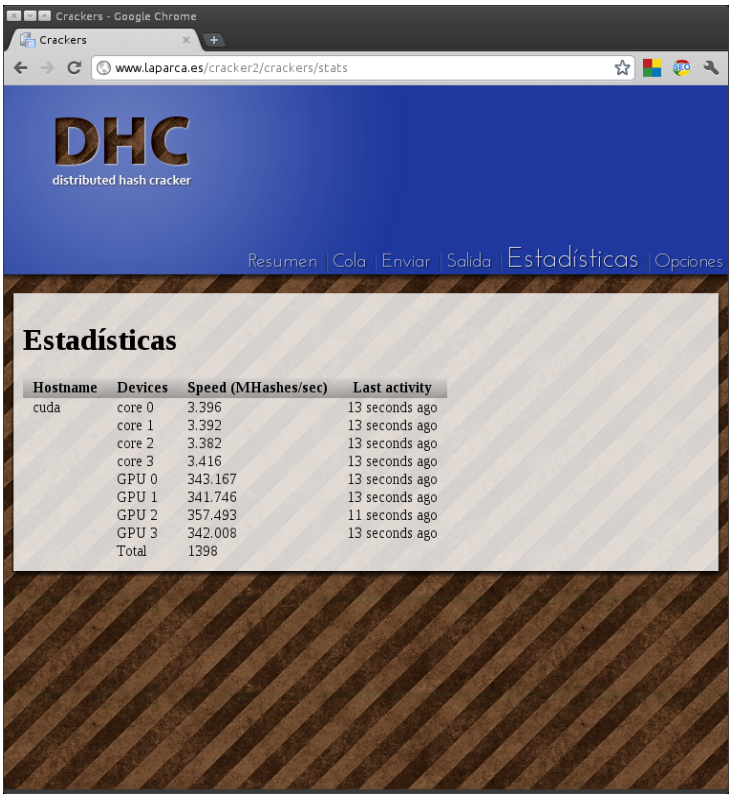

Figura C.5: *Estadísticas de funcionamiento*

seguirán los siguientes pasos en el agente sin olvidar que tras la modificación hay que reiniciar el agente.

C.3.1. Instalación de nuevos plugins

Para instalar nuevos *plugins* en el sistema solo hay que copiar estos al directorio /usr/lib/cracker/ptx. Los *plugins* tienen todos la extensión .aplug.so.

C.3.2. Eliminación plugins obsoletos o erróneos

Para eliminar un *plugin* con errores u obsoleto basta con borrar el fichero que lo contiene:

```
$ sudo rm /usr/local/cracker/ptr/nombre_plugin.aplug.so
```

C.4. Configuraciones avanzadas

En ciertas ocasiones puede hacer falta realizar instalaciones especiales de DHC. En esta sección se van a explicar dos opciones que hemos detectado.

C.4.1. Varios controladores con un MySQL

Puede darse el caso de que se necesite disponer de varios controladores de DHC pero que compartan una única base de datos. Esto puede resultar útil si la carga sobre un único controlador es muy elevada.

Para llevar a cabo este proceso se deben instalar dos o más controladores, solo con servidor web siguiendo los pasos descritos anteriormente, y a parte se configurará otro servidor con MySQL. Ambos controladores se configurarán para utilizar dicho servidor de MySQL.

Una vez que se inicien los agentes se irá pasando a cada uno la URL del controlador que deseemos o podemos configurar un balanceador de carga automático.

C.4.2. Cambio de los tiempos de espera para evitar exceso de reciclado de WU

En ciertas ocasiones, los agentes pueden tardar mucho en devolver el resultado de una WU por lo que se hace necesario modificar los tiempo de

caducidad para evitar un exceso de reciclados que ralentizarían sobremedida el proceso de comprobación.

C.4.3. Cambio de los directorios de búsqueda predeterminados

Cuando un agente se inicia siempre comprueba la existencia de *plugins* en unos directorios predefinidos. Además, también busca en estos mismos directorios los ficheros binarios de CUDA que le puedan ser solicitados por los algoritmos. Por defecto estos directorios son: el directorio desde el que se ejecuta el agente (directorio ./), el directorio *ptx* desde el que se ejecuta el agente (directorio ./ptx) y el directorio */usr/lib/cracker/ptx*.

En caso de que se quisiese cambiar este comportamiento se puede editar el fichero config.cpp del agente para añadir o eliminar directorios de búsqueda.

Apéndice D

Glosario

Algoritmo Para el propósito de este proyecto un algoritmo es cualquier código destinado a poder ejecutarse en CPU o GPU con el fin de obtener una clave a partir de un conjunto de datos cifrados.

API Aplication Programming Interface o interfaz de programación de aplicación. Es un conjunto de especificaciones destinadas a facilitar la realización de ciertas tareas.

Colision Una colisión en el ámbito de las funciones resumen es cuando se dispone de dos mensajes diferentes que producen el mismo resumen.

Commit Cada vez que se realizan modificaciones sobre fichero y se desea almacenar estas en el repositorio de versiones se realiza una operación de *commit*.

CPU *Central Process Unit*. Es la unidad central de proceso y está encargada de ejecutar el código binario.

Distribución Linux En el mundo del sistema operativo Linux se llama distribución a un paquete que contiene el núcleo Linux junto a un determinado conjunto de herramientas.

Fetch Herramienta de git que permite consultar a un repositorio remoto por los cambios que han ocurrido. Estos cambios no se aplican localmente.

Framework Conjunto de herramientas y especificaciones que permiten desarrollar aplicaciones de forma sencilla. La diferencia con el API es que

esta última solo define bibliotecas para utilizar, mientras que el framework puede ir acompañado de herramientas.

GPU *Graphic Process Unit* o unidad de proceso gráfico es el procesador utilizado por una tarjeta gráfica para controlar las operaciones que se realizan sobre la misma. En la actualidad permiten la carga de programas de usuario especialmente diseñados.

HEAD En git HEAD hace referencia a la última revisión que se encuentra en el repositorio.

Linux Es un núcleo de sistema operativo que nació del núcleo Minix.

Makefile Es un fichero que tiene las reglas a seguir para construir (compilar) un proyecto software.

MVC Patrón arquitectónico Modelo-Vista-Controlador. Es un modo de organizar el código de tal forma que se diferencia claramente qué se encarga de la persistencia de datos (el modelo), qué se encarga del control de la lógica del programa (el controlador) y qué se encarga de presentar los datos al usuario (la vista).

Núcleo En sistemas operativos el núcleo es un software que se encarga de facilitar los recursos del sistema a los programas.

Pull En git se utiliza pull para aplicar a la rama actual de desarrollo que se encuentre activa los últimos cambios que se hayan hecho. Es recomendable hacer *fetch* previamente.

Rack Es un armario especial para poder poner equipos informáticos como ordenadores, switch, etc.

Rama de desarrollo Es una línea de desarrollo de un proyecto en la que se hacen modificaciones. Las modificaciones realizadas en una rama no son vistas por el resto de ramas.

Push Es el mandato que utiliza git para enviar todos los cambios locales a un repositorio remoto. Si no se indica ningún repositorio se utilizará el repositorio *origin*.

URL Dirección de un elemento en internet. Suele estar definido por el tipo de protocolo, la dirección de la máquina y la ruta al elemento. Por ejemplo, http://www.google.com/reader. En este caso se accede a la máquina www.elpais.es utilizando el protocolo HTTP para solicitar /reader.

Apéndice E

Códigos destacados

En este apéndice se van a presentar los ficheros de código fuente que probablemente sean más interesantes en el proyecto. La mayoría representan únicamente aquellos ficheros que disponen de código importante sobre los cambios realizados.

Por otra parte se presenta también la organización de los ficheros para que en caso de acceder al código se sepa dónde se encuentra cada cosa.

E.1. Organización del código

El código fuente del proyecto se encuentra en la subcarpeta v3. En esta se puede encontrar la carpeta del agente, del controlador y la documentación:

v3/agent/ Código fuente del agente.

v3/controller/ Contiene el código fuente del controlador.

v3/controller/components/ Aquí se encuentran herramientas que son de utilidad para el desarrollo de la aplicación.

v3/controller/app/config/ Almacena la información de configuración de la base de datos y de algunas variables del controlador.

v3/controller/app/models En esta carpeta se encuentra la definición del modelo de base de datos. Cada fichero representa una tabla.

v3/controller/app/views La representación de los datos devueltos, tanto para la web como para los agentes, se lleva a cabo desde este punto.

v3/controller/app/controllers/ El control de las acciones se lleva a cabo en esta carpeta.

v3/doc Código fuente de esta memoria.

E.2. Códigos importantes del agente

E.2.1. v3/agent/config.cpp

```
1  /*****************************************************************************
2  *                                                                            *
3  * Distributed Hash Cracker v3.0                                              *
4  *                                                                            *
5  * Copyright (c) 2009 RPISEC.                                                 *
6  * Copyright (C) 2010 Samuel Rodriguez Sevilla                                *
7  * All rights reserved.                                                       *
8  *                                                                            *
9  * Redistribution and use in source and binary forms, with or without modifi- *
10 * cation, are permitted provided that the following conditions are met:      *
11 *                                                                            *
12 *     * Redistributions of source code must retain the above copyright notice *
13 *       this list of conditions and the following disclaimer.                *
14 *                                                                            *
15 *     * Redistributions in binary form must reproduce the above copyright    *
16 *       notice, this list of conditions and the following disclaimer in the  *
17 *       documentation and/or other materials provided with the distribution. *
18 *                                                                            *
19 *     * Neither the name of RPISEC nor the names of its contributors may be  *
20 *       used to endorse or promote products derived from this software without *
21 *       specific prior written permission.                                   *
22 *                                                                            *
23 * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26 * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33 *                                                                            *
34 *****************************************************************************/
35
36 #include "debug.h"
37 #include <string>
38 #include <vector>
39 using namespace std;
40
41 /*!
42  * @file config.cpp
43  * @brief This file sets de values of global variables
44  */
45
46 /* create unused names */
47 #define UNUSED UNUSED_(__COUNTER__)
48 #define UNUSED_(counter) UNUSED__(counter)
49 #define UNUSED__(counter) UNUSED_ ## counter
50
51 /* register a path inside the ConfigDirectories vector */
52 #define REGISTER_CONFIG_PATH(path) static _register_config_path UNUSED(path)
53
54
55 /*!
56  * @brief List of directories where the cracker looks for plugins and other files
57  */
58 vector<string> ConfigDirectories;
```

```
59
60  /*!
61   * @brief Used to register paths inside ConfigDirectories
62   */
63  struct _register_config_path {
64    _register_config_path(string str) {
65      ConfigDirectories.push_back(str);
66    }
67  };
68
69  REGISTER_CONFIG_PATH("./");
70  REGISTER_CONFIG_PATH("ptx/");
71  REGISTER_CONFIG_PATH("/usr/lib/cracker/ptx/");
72
73  /*!
74   * @brief Sets the debug level if debug is active
75   */
76  INIT_LOG(DEBUG);
```

E.2.2. v3/agent/Algorithm.h

```
1   /*****************************************************************************
2   *                                                                            *
3   * Distributed Hash Cracker v3.0                                              *
4   *                                                                            *
5   * Copyright (c) 2009 RPISEC.                                                 *
6   * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
7   * All rights reserved.                                                       *
8   *                                                                            *
9   * Redistribution and use in source and binary forms, with or without modifi- *
10  * cation, are permitted provided that the following conditions are met:      *
11  *                                                                            *
12  *     * Redistributions of source code must retain the above copyright notice *
13  *       this list of conditions and the following disclaimer.                *
14  *                                                                            *
15  *     * Redistributions in binary form must reproduce the above copyright    *
16  *       notice, this list of conditions and the following disclaimer in the  *
17  *       documentation and/or other materials provided with the distribution. *
18  *                                                                            *
19  *     * Neither the name of RPISEC nor the names of its contributors may be  *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                                   *
22  *                                                                            *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33  *                                                                            *
34  *****************************************************************************/
35
36
37  /*!
38   *   @file Algorithm.h
39   *
40   *   @brief Declararation of an algorithm functionality.
41   */
42
43  #ifndef ALGORITHM_H
44  #define ALGORITHM_H
45
46  #include <string>
47  #include <vector>
48  #include "WorkUnit.h"
49  #include "agent.h"
50  #include "LowLevel.h"
51
52  using namespace std;
53
54
```

```
55  /*!
56   *  @class Algorithm
57   *  @brief Algorithm represents a cryptographic function the system can use to
58   *  test password fortress.
59   *
60   *  When Distributed Hash Cracker test a Hash, this hash have to be procesed
61   *  using a concrete algorithm like MD5, SHA256, etc. Algorithms are the
62   *  central functionallity of Distributed Hash Cracer and they are created
63   *  for optimize the old code form version 3.0 where the algorithm
64   *  functionallity where mixed with between them doing the code hard to read.
65   */
66  class Algorithm
67  {
68  public:
69    /*!
70     *  @brief Returns the name of the algorithm
71     *  @return The algorithm name
72     */
73    virtual string GetName() = 0;
74    /*!
75     *  @brief Returns the length of a hash generated by the algorithm
76     *  @return The generated hash length
77     */
78    virtual int  HashLength() = 0;
79    /*!
80     *  @brief Each algorithm requires an input (normally a hash) with
81     *  a concrete length. This method returns the length of this input.
82     *  @return The algorithm input length
83     */
84    virtual int  InputLength() = 0;
85    /*!
86     *  @brief Executes the CPU version of the algorithm. It is useful
87     *  if the system is not GPU accelerated.
88     *  @param wu Work unit to process.
89     *  @param nCore CPU core used to process the work unit.
90     */
91    virtual void ExecuteCPU(WorkUnit& wu, int nCore) = 0;
92  #ifdef CUDA_ENABLED
93    /*!
94     *  @brief Executes the GPU version of the algorithm.
95     *  @param wu Work unit to process.
96     *  @param pDevice Information about the GPU device.
97     *  @param pContext Information about the CUDA device context.
98     */
99    virtual void ExecuteGPU(WorkUnit& wu, Device* pDevice, CudaContext* pContext) = 0;
100 #endif
101   /*!
102    *  @brief Returns if the algorithm can use GPU capabilities
103    *  @return Tue if the Algorithm can use a GPU
104    */
105   virtual bool IsGPUCapable() = 0;
106   /*!
107    *  @brief Returns if the algorithm cas use GPU capabilites
108    *  @return True if the algorithm can use a CPU
109    */
110   virtual bool IsCPUCapable() = 0;
111   /*!
112    *  @brief Number of registers used by the GPU version of the algorithm.
113    *  This value can be obtained using the cubin compilation of the GPU unit. If it is
114    *  unknown it returns 0.
115    *  @return The number of registers used
116    */
117   virtual int RegistersUsed() { return 0; }
118   /*!
119    *  @brief Amount of constant memory used by the GPU version of the algorithm.
120    *  This value can be obtained using the cubin compilation of the GPU unit. If it is
121    *  unknown it return 0.
122    *  @return The amount of constant memory used
123    */
124   virtual int ConstantMemoryUsed() { return 0; }
125   /*!
126    *  @brief Amount of shared memory used by the GPU version of the algorithm.
127    *  This value can be obtained using the cubin compilation of the GPU unit. If it is
128    *  unknown it return 0.
129    *  @return The amount of shared memory used
130    */
131   virtual int SharedMemoryUsed() { return 0; }
```

```
132 };
133
134 #endif
```

E.2.3. v3/agent/AlgorithmFactory.h

```
1  /*****************************************************************************
2  *                                                                            *
3  * Distributed Hash Cracker v3.0                                              *
4  *                                                                            *
5  * Copyright (c) 2009 RPISEC.                                                 *
6  * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
7  * All rights reserved.                                                       *
8  *                                                                            *
9  * Redistribution and use in source and binary forms, with or without modifi- *
10 * cation, are permitted provided that the following conditions are met:      *
11 *                                                                            *
12 *     * Redistributions of source code must retain the above copyright notice *
13 *       this list of conditions and the following disclaimer.                *
14 *                                                                            *
15 *     * Redistributions in binary form must reproduce the above copyright    *
16 *       notice, this list of conditions and the following disclaimer in the  *
17 *       documentation and/or other materials provided with the distribution. *
18 *                                                                            *
19 *     * Neither the name of RPISEC nor the names of its contributors may be  *
20 *       used to endorse or promote products derived from this software without *
21 *       specific prior written permission.                                   *
22 *                                                                            *
23 * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26 * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33 *                                                                            *
34 *****************************************************************************/
35
36
37 /*!
38  *  @file Algorithm.h
39  *
40  *  @brief Declares the algorith factory facility.
41  *
42  *  Algorithms are the essential functionallity of the agent. They do the
43  *  real work of processing the data suministrated by the controller.
44  */
45
46 #ifndef ALGORITHM_FACTORY_H
47 #define ALGORITHM_FACTORY_H
48
49 #include "Algorithm.h"
50
51 /*!
52  *  @def INIT_ALGORITHMS
53  *  @brief Initializes the algorithm system.
54  *
55  *  This method/macro has to be used in the global section of a source file
56  *  for initilize the internal algorithm system.
57  */
58 #define INIT_ALGORITHMS() AlgorithmFactory::algorithm_list AlgorithmFactory::vAlgorithms
59
60
61 /*!
62  *  @class AlgorithmFactory
63  *  @brief AlgorithmFactory is the utility to manage algorithms.
64  *
65  *  With AlgorithmFactory we can instance new algorithms.
66  */
67 class AlgorithmFactory
68 {
69 /*!
```

```
70    *  \defgroup Algorithm administration system
71    *  \page Algorithm Organization of algorithm
72    *
73    *  Algorithms can be dynamic loaded or can be registered in compilation time
74    *  and to achieve this purpose a algorithm management system is needed.
75    */
76   /*@{*/
77   public:
78     typedef vector<Algorithm *> algorithm_list;
79     typedef vector<Algorithm *>::iterator algorithm_iterator;
80
81   private:
82     /*!
83      *  @brief The list of algorithms registered in the system.
84      *
85      *  Stores the list of algorithms registered in the system for grand
86      *  the hability of query them. This attribute has to be initialized in
87      *  one source file to be accesible. For that initialization you have to
88      *  declare it as follows:
89      *  Algorithm::algorithm_list Algorithm::vAlgorithms;
90      *  You can also use the macro INIT_ALGORITHMS()
91      */
92     static algorithm_list vAlgorithms;
93
94   public:
95     /*!
96      *  @brief Returns the list of algorithms
97      *
98      *  When a functionallity that implies algorithms is need but this
99      *  functionallity do not exists a this method help us. It returns the
100     *  list of algorithms and then it can be used to do the desire
101     *  functionallity.
102     *
103     *  @return The algorithm list
104     */
105    static algorithm_list& GetAlgorithmList()
106    {
107      return vAlgorithms;
108    }
109
110    /*!
111     *  @brief Register an algorithm in a internal list used for
112     *  control the algorithms.
113     *  @param pAlgorithm The algorithm to register
114     */
115    static void RegisterAlgorithm(Algorithm *pAlgorithm)
116    {
117      vAlgorithms.push_back(pAlgorithm);
118    }
119    /*!
120     *  @brief Returns the algorithm whos name is the user specified
121     *  @param name The algorithm searched name
122     *  @return The algorithm or NULL if it does not exists
123     */
124    static Algorithm *GetAlgorithm(const string& name)
125    {
126      algorithm_iterator end = vAlgorithms.end();
127      for(algorithm_iterator it = vAlgorithms.begin(); it != end; it++)
128      {
129        if((*it)->GetName() == name)
130          return *it;
131      }
132      return NULL;
133    }
134    /*!
135     *  @brief Return the name of the registered algorithms
136     *  @return The list of names
137     */
138    static vector<string> GetAlgorithmNames();
139    /*!
140     *  @brief Returns the list of algorithms that matches with the
141     *  function func.
142     *
143     *  There are lots of situations where the normal functions are not
144     *  suficient to obtain the desire algorithm. To solve this problem
145     *  GetAlgoritms function use a function or a functor to look inside
146     *  each algorithm and to select each algorithm needed. The functor
```

```
147     *  only has to implements the () operator with one parameter of
148     *  Algorithm * type. This functor o functions has to return true
149     *  if the algorithm is selected and false in other case.
150     *
151     *  @param func Function or functor needed to get the algorithms
152     *  @return The algorithm list.
153     */
154    template<typename Func>
155    static algorithm_list GetAlgorithms(Func func)
156    {
157      algorithm_list result;
158      algorithm_iterator end = vAlgorithms.end();
159      for(algorithm_iterator it = vAlgorithms.begin(); it != end; it ++)
160        if(func(*it))
161          result.push_back(*it);
162      return result;
163    }
164    /*!
165     * @brief Test internal functionallity.
166     */
167    static void Test();
168    /*@}*/
169  };
170
171  #endif
```

E.2.4. v3/agent/AlgorithmFactory.cpp

```
1   /*****************************************************************************
2   *                                                                            *
3   * Distributed Hash Cracker v3.0                                              *
4   *                                                                            *
5   * Copyright (c) 2009 RPISEC.                                                 *
6   * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
7   * All rights reserved.                                                       *
8   *                                                                            *
9   * Redistribution and use in source and binary forms, with or without modifi- *
10  * cation, are permitted provided that the following conditions are met:      *
11  *                                                                            *
12  *     * Redistributions of source code must retain the above copyright notice *
13  *       this list of conditions and the following disclaimer.                *
14  *                                                                            *
15  *     * Redistributions in binary form must reproduce the above copyright    *
16  *       notice, this list of conditions and the following disclaimer in the  *
17  *       documentation and/or other materials provided with the distribution. *
18  *                                                                            *
19  *     * Neither the name of RPISEC nor the names of its contributors may be  *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                                   *
22  *                                                                            *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33  *                                                                            *
34  *****************************************************************************/
35  #include "AlgorithmFactory.h"
36  #include <iostream>
37
38  /*!
39   * @brief Test internal functionallity.
40   *
41   * For test the AlgorithmFactory it shows each algorithm registered and it
42   * tries to access to some functions.
43   */
44  void AlgorithmFactory::Test()
45  {
46    cout << "Showing the entire algortihm list" << endl;
47    for(algorithm_iterator it = vAlgorithms.begin(); it != vAlgorithms.end(); it++)
```

```
48    {
49      Algorithm *alg = *it;
50
51      cout << "Name: " << alg->GetName() << endl;
52      cout << "   Is CUDA capable: " << (alg->IsGPUCapable()? "yes" : "no") << endl;
53      cout << "   Is CPU capable : " << (alg->IsCPUCapable()? "yes" : "no") << endl;
54      cout << "   Hash length    : " << alg->HashLength() << endl;
55    }
56  }
57
58  /*!
59   *  @brief Return the name of the registered algorithms
60   *  @return The list of names
61   */
62  vector<string> AlgorithmFactory::GetAlgorithmNames()
63  {
64    vector<string> v;
65    for(algorithm_iterator it = vAlgorithms.begin(); it != vAlgorithms.end(); it++)
66    {
67      v.push_back((*it)->GetName());
68    }
69    return vector<string>(v);
70  }
```

E.2.5. v3/agent/Executor.h

```
1   /*****************************************************************************
2   *                                                                            *
3   * Distributed Hash Cracker v3.0                                              *
4   *                                                                            *
5   * Copyright (c) 2009 RPISEC.                                                 *
6   * Copyright (C) 2010 Samuel Rodriguez Sevilla                                *
7   * All rights reserved.                                                       *
8   *                                                                            *
9   * Redistribution and use in source and binary forms, with or without modifi- *
10  * cation, are permitted provided that the following conditions are met:      *
11  *                                                                            *
12  *     * Redistributions of source code must retain the above copyright notice *
13  *       this list of conditions and the following disclaimer.                *
14  *                                                                            *
15  *     * Redistributions in binary form must reproduce the above copyright    *
16  *       notice, this list of conditions and the following disclaimer in the  *
17  *       documentation and/or other materials provided with the distribution. *
18  *                                                                            *
19  *     * Neither the name of RPISEC nor the names of its contributors may be  *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                                   *
22  *                                                                            *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33  *                                                                            *
34  *****************************************************************************/
35  #ifndef EXECUTOR_H
36  #define EXECUTOR_H
37
38  #include "Algorithm.h"
39  #include <map>
40  #include <string>
41  using namespace std;
42
43  /*!
44   *  @def GET_EXECUTOR_PARAM
45   *  @brief This define is used as parameter for the SelectAlgorithms
46   *  class and indicates that it has select algorithms that use GPU.
47   */
48  #define GET_EXECUTOR_PARAM(type, name)                                  \
49  type name;                                                              \
```

```
50  do                                                                    \
51  {                                                                     \
52     executor_parameters_iterator value = parameters.find(#name);       \
53     if(value != parameters.end())                                      \
54     {                                                                  \
55        name = *(static_cast<type*>(value->second));                    \
56     }                                                                  \
57     else                                                               \
58        throw "Parameter is not defined";                               \
59  }while(0)
60
61
62  typedef map<string, void*> executor_parameters;
63  typedef map<string, void*>::iterator executor_parameters_iterator;
64
65  /*!
66   * @class Executor
67   * @brief Executor prepares the GPU for run Algorithms using a known method.
68   *
69   * When an Algorithm must be executed in a GPU this algorithm needs its values
70   * loaded in the GPU memory. An executor represents a mechanism to initialize
71   * the GPU for an algorithm.
72   */
73  class Executor
74  {
75  public:
76  #ifdef CUDA_ENABLED
77    /*!
78     * @brief Executes an Algorithm into the GPU
79     */
80    virtual void Execute(Algorithm *alg, WorkUnit& wu, Device* pDevice, CudaContext* pContext, executor_parameters& param
81    void operator()(Algorithm *alg, WorkUnit& wu, Device* pDevice, CudaContext* pContext, executor_parameters& parameters)
82    {
83      Execute(alg, wu, pDevice, pContext, parameters);
84    }
85  #endif
86    /*!
87     * @brief Returns the Executor name
88     * @return The name of the Executor
89     */
90    virtual string GetName() = 0;
91  };
92
93
94  #endif
```

E.2.6. v3/agent/ExecutorFactory.h

```
1   /*****************************************************************************
2   *                                                                            *
3   * Distributed Hash Cracker v3.0                                              *
4   *                                                                            *
5   * Copyright (c) 2009 RPISEC.                                                 *
6   * Copyright (C) 2010 Samuel Rodriguez Sevilla
7   * All rights reserved.                                                       *
8   *                                                                            *
9   * Redistribution and use in source and binary forms, with or without modifi- *
10  * cation, are permitted provided that the following conditions are met:      *
11  *                                                                            *
12  *     * Redistributions of source code must retain the above copyright notice *
13  *       this list of conditions and the following disclaimer.                *
14  *                                                                            *
15  *     * Redistributions in binary form must reproduce the above copyright    *
16  *       notice, this list of conditions and the following disclaimer in the  *
17  *       documentation and/or other materials provided with the distribution. *
18  *                                                                            *
19  *     * Neither the name of RPISEC nor the names of its contributors may be  *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                                   *
22  *                                                                            *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
```

```
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR      *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF      *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING        *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS          *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33  *                                                                           *
34  *****************************************************************************/
35
36  #ifndef EXECUTOR_FACTORY_H
37  #define EXECUTOR_FACTORY_H
38
39  #include "Executor.h"
40  #include <string>
41  #include <map>
42  using namespace std;
43
44  /*!
45   *  @def INIT_EXECUTORS
46   *  @brief Initializes the executors system.
47   *
48   *  This method/macro has to be used in the global section of a source file
49   *  for initilize the internal executors system.
50   */
51  #define INIT_EXECUTORS() map<string, Executor *> ExecutorFactory::vExecutors
52
53  /*!
54   */
55  class ExecutorFactory
56  {
57  private:
58    static map<string, Executor *> vExecutors;
59
60  public:
61    static Executor* Get(const string& name);
62    static void Register(Executor *ex);
63  };
64
65  #endif
```

E.2.7. v3/agent/ExecutorFactory.cpp

```
1  #include "ExecutorFactory.h"
2
3  Executor* ExecutorFactory::Get(const string& name)
4  {
5    map<string, Executor*>::iterator value = vExecutors.find(name);
6    if(value == vExecutors.end())
7    {
8      throw "Executor does not exist";
9    }
10
11   return value->second;
12 }
13
14 void ExecutorFactory::Register(Executor *ex)
15 {
16   vExecutors[ex->GetName()] = ex;
17 }
```

E.2.8. v3/agent/Plugin.h

```
1  /*****************************************************************************
2  *                                                                           *
3  * Distributed Hash Cracker v3.0                                             *
4  *                                                                           *
5  * Copyright (c) 2009 RPISEC.                                                *
6  * Copyright (C) 2010 Samuel Rodriguez Sevilla                               *
7  * All rights reserved.                                                      *
8  *                                                                           *
9  * Redistribution and use in source and binary forms, with or without modifi- *
10 * cation, are permitted provided that the following conditions are met:     *
11 *                                                                           *
```

```
12  *     * Redistributions of source code must retain the above copyright notice  *
13  *       this list of conditions and the following disclaimer.                  *
14  *                                                                              *
15  *     * Redistributions in binary form must reproduce the above copyright      *
16  *       notice, this list of conditions and the following disclaimer in the    *
17  *       documentation and/or other materials provided with the distribution.   *
18  *                                                                              *
19  *     * Neither the name of RPISEC nor the names of its contributors may be    *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                                     *
22  *                                                                              *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED       *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF         *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN      *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,   *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED     *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR       *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF       *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING         *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS           *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.                 *
33  *                                                                              *
34  *******************************************************************************/
35  #ifndef __PLUGIN_H__
36  #define __PLUGIN_H__
37
38  #include <string>
39  #include <vector>
40  using namespace std;
41
42
43  /*!
44   *  @file Plugin.h
45   *
46   *  @brief Plugin facilities.
47   */
48
49  #define EXPORT extern "C"
50
51  #define BEGIN_PLUGIN(author) \
52  EXPORT  PluginFactory* GetPluginFactory() {\
53    PluginFactory *rtn = new PluginFactory(author); \
54
55  #define END_PLUGIN() \
56    return rtn; \
57  }
58
59  #define ADD_ALGORITHM(class, name, version) \
60    rtn->AddFacility(new PluginFacilityAlgorithm<class>(name, version));
61  #define ADD_EXECUTOR(class, name, version) \
62    rtn->AddFacility(new PluginFacilityExecutor<class>(name, version));
63
64  #define FACILITY_ALGORITHM 0
65  #define FACILITY_EXECUTOR  1
66
67
68  /*!
69   * @class Pluginfacility
70   * @brief Defines information about a plugin facility and it can instance one.
71   *
72   * A plugin facility is an algorithm or an executor that can be dinamically
73   * loaded by the agent.
74   */
75  class PluginFacility {
76    string        facility_name;
77    int           facility_type;
78    unsigned int facility_version;
79  public:
80    PluginFacility(string name, int type, unsigned int version) :
81    facility_name(name), facility_type(type), facility_version(version)
82    {}
83    /*!
84    * @brief returns the facility version
85    */
86    unsigned int GetVersion() { return facility_version; }
87    /*!
88     * @brief returns the type of the facility. May be FACILITY_ALGORITHM or FACILITY_EXECUTOR.
```

```
89      */
90    int           GetType() { return facility_type; }
91    /*!
92     * @brief returns the facility name.
93     */
94    string        GetName() { return facility_name; }
95    /*!
96     * @brief returns a new instance of a facility.
97     */
98    virtual void* GetInstance() = 0;
99  };
100
101 template<typename Algorithm>
102 class PluginFacilityAlgorithm : public PluginFacility {
103 public:
104   PluginFacilityAlgorithm(string name, unsigned int version) :
105   PluginFacility(name, FACILITY_ALGORITHM, version)
106   /*facility_name(name), facility_type(FACILITY_ALGORTHM), facility_version(verion)*/
107   {}
108
109   void* GetInstance() { return new Algorithm(); }
110 };
111
112 template<typename Executor>
113 class PluginFacilityExecutor : public PluginFacility {
114 public:
115   PluginFacilityExecutor(string name,  unsigned int version) :
116   PluginFacility(name, FACILITY_EXECUTOR, version)
117   /*facility_name(name), facility_type(FACILITY_EXECUTOR), facility_version(verion)*/
118   {}
119
120   void* GetInstance() { return new Executor(); }
121 };
122
123 /*!
124  * @class PluginFactory
125  * @brief Used to load the facilities inside a plugin.
126  *
127  * When a Plugin is loaded automatically the agent intance a PluginFactory
128  * calling the GetPluginFactory method.
129  */
130 class PluginFactory {
131   vector<PluginFacility*> facilites;
132   string _author;
133 public:
134   /*!
135    * @brief When a PluginFactory is craeted an author must be provided for identify the plugin.
136    */
137   PluginFactory(string author) : _author(author)
138   {}
139   /*!
140    * @brief returns plugin author name.
141    */
142   string GetAuthorName()
143   {
144     return _author;
145   }
146
147   /*!
148    * @brief returns a list with all the facilities defined in the plugin.
149    */
150   vector<PluginFacility*> GetFacilities()
151   {
152     return facilites;
153   }
154
155   /*!
156    * @brief add a facility to the plugin list.
157    */
158   void AddFacility(PluginFacility *facility)
159   {
160     facilites.push_back(facility);
161   }
162 };
163
164 #endif
```

E.2.9. v3/agent/PluginLoader.h

```
1  /*****************************************************************************
2  *                                                                            *
3  * Distributed Hash Cracker v3.0                                              *
4  *                                                                            *
5  * Copyright (c) 2009 RPISEC.                                                 *
6  * Copyright (C) 2010 Samuel Rodriguez Sevilla                                *
7  * All rights reserved.                                                       *
8  *                                                                            *
9  * Redistribution and use in source and binary forms, with or without modifi- *
10 * cation, are permitted provided that the following conditions are met:      *
11 *                                                                            *
12 *     * Redistributions of source code must retain the above copyright notice *
13 *       this list of conditions and the following disclaimer.                *
14 *                                                                            *
15 *     * Redistributions in binary form must reproduce the above copyright    *
16 *       notice, this list of conditions and the following disclaimer in the  *
17 *       documentation and/or other materials provided with the distribution. *
18 *                                                                            *
19 *     * Neither the name of RPISEC nor the names of its contributors may be  *
20 *       used to endorse or promote products derived from this software without *
21 *       specific prior written permission.                                   *
22 *                                                                            *
23 * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26 * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33 *                                                                            *
34 *****************************************************************************/
35 #ifndef __PLUGIN_LOADER_H__
36 #define __PLUGIN_LOADER_H__
37
38 #include <string>
39 using namespace std;
40
41 /*!
42  * @file PluginLoader.h
43  *
44  * @brief Adds methods for plugin load.
45  */
46
47 /* TODO añadir metodos para buscar todos los ficheros posibles que sean plugin.
48  * TODO cargar cada plugin e instanciar sus facilidades.
49  */
50 /**
51  * @class PluginLoader
52  * @brief Finds and loads the plugins installed on the predefined directories
53  *
54  * To load a plugin requiere, first of all, look into all the posible
55  * directories where they can be allocated (the list of directories is the global
56  * variable @ConfigDirectories@ in config.cpp).
57  *
58  * Each plugin file has the .aplug.so and is loaded using ldopen (UNIX/Linux
59  * only for now).
60  */
61 class PluginLoader
62 {
63 public:
64   /**
65    * @brief Loads all the plugins installed
66    */
67   static void Load();
68
69 private:
70   /**
71    * @brief Try to load the plugin in the file @file@
72    */
73   static void Load(string file);
74 };
```

```
75
76 #endif
```

E.2.10. v3/agent/PluginLoader.cpp

```
1  /*****************************************************************************
2  *                                                                            *
3  * Distributed Hash Cracker v3.0                                              *
4  *                                                                            *
5  * Copyright (c) 2009 RPISEC.                                                 *
6  * Copyright (C) 2010 Samuel Rodriguez Sevilla                                *
7  * All rights reserved.                                                       *
8  *                                                                            *
9  * Redistribution and use in source and binary forms, with or without modifi- *
10 * cation, are permitted provided that the following conditions are met:      *
11 *                                                                            *
12 *     * Redistributions of source code must retain the above copyright notice *
13 *       this list of conditions and the following disclaimer.                *
14 *                                                                            *
15 *     * Redistributions in binary form must reproduce the above copyright    *
16 *       notice, this list of conditions and the following disclaimer in the  *
17 *       documentation and/or other materials provided with the distribution. *
18 *                                                                            *
19 *     * Neither the name of RPISEC nor the names of its contributors may be  *
20 *       used to endorse or promote products derived from this software without *
21 *       specific prior written permission.                                   *
22 *                                                                            *
23 * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26 * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33 *                                                                            *
34 *****************************************************************************/
35 #include "Plugin.h"
36 #include "PluginLoader.h"
37 #include "AlgorithmFactory.h"
38 #include "ExecutorFactory.h"
39 #include "debug.h"
40 #include <dlfcn.h>
41 #include <sys/types.h>
42 #include <dirent.h>
43 #include <vector>
44 #include <string>
45
46 using namespace std;
47
48 extern vector<string> ConfigDirectories;
49
50 void PluginLoader::Load()
51 {
52   DO_ENTER("PluginLoader", "Load");
53
54   for (vector<string>::iterator dir_it = ConfigDirectories.begin() ; dir_it != ConfigDirectories.end(); ++dir
55   {
56     DIR* dir;
57     struct dirent* ent;
58
59     DO_MESSAGE(string("Opening ") << *dir_it);
60     dir = opendir(dir_it->c_str());
61
62     /* No directory? Try next */
63     if(dir == NULL) continue;
64
65     while((ent = readdir(dir)) != NULL)
66     {
67       string file_name(ent->d_name);
68       string file_ext(".aplug.so");
69       if(file_name.size() > file_ext.size() && file_name.substr(file_name.size() - file_ext.size(), file_ext.
70       {
```

```
71            PluginLoader::Load(*dir_it + file_name);
72          }
73        }
74        closedir(dir);
75      }
76    }
77
78    typedef PluginFactory* ptrPluginFactory;
79    typedef ptrPluginFactory (*GetPluginFactory_t)();
80
81    void PluginLoader::Load(string file)
82    {
83      DO_ENTER("PluginLoader", "Load");
84      DO_MESSAGE(string("Try to load ") << file);
85      void *handle;
86
87      handle = dlopen(file.c_str(), RTLD_NOW );
88      if(handle == NULL) {
89        DO_MESSAGE(string("Library cannot be opened"));
90        return;
91      }
92
93      void *func = dlsym(handle, "GetPluginFactory");
94      if(func == NULL)
95      {
96        DO_MESSAGE(string("No GetPluginFactory in plugin"));
97      }
98      else
99      {
100       DO_MESSAGE(string("GetPluginFactory found! :-)"));
101
102       GetPluginFactory_t GetPluginFactory = (GetPluginFactory_t)func;
103
104       PluginFactory* factory = GetPluginFactory();
105
106       DO_MESSAGE(string("Author name: ") << factory->GetAuthorName());
107       vector<PluginFacility*> facilities = factory->GetFacilities();
108
109       for(vector<PluginFacility*>::iterator it = facilities.begin(); it != facilities.end(); ++it)
110       {
111         DO_MESSAGE(string("Facility name    : ") << (*it)->GetName());
112         DO_MESSAGE(string("Facility version : ") << (*it)->GetVersion());
113         DO_MESSAGE(string("Facility type    : ") << ((*it)->GetType() == FACILITY_ALGORITHM? "Algorithm" : "Executor"));
114
115         switch((*it)->GetType())
116         {
117           case FACILITY_ALGORITHM:
118             AlgorithmFactory::RegisterAlgorithm((Algorithm *)(*it)->GetInstance());
119             break;
120           case FACILITY_EXECUTOR:
121             ExecutorFactory::Register((Executor *)(*it)->GetInstance());
122             break;
123         }
124       }
125
126       delete factory;
127     }
128     /* The handle must be opened because if it gets closed frees the plugin
129      * memory and then it becomes unreachable.
130      */
131     //dlclose(handle);
132   }
```

E.2.11. v3/agent/debug.h

```
1   /*****************************************************************************
2   *                                                                            *
3   * Distributed Hash Cracker v3.0                                              *
4   *                                                                            *
5   * Copyright (c) 2009 RPISEC.                                                 *
6   * Copyright (C) 2010 Samuel Rodriguez Sevilla                                *
7   * All rights reserved.                                                       *
8   *                                                                            *
9   * Redistribution and use in source and binary forms, with or without modifi- *
10  * cation, are permitted provided that the following conditions are met:      *
```

```
11  *                                                                      *
12  *     * Redistributions of source code must retain the above copyright notice  *
13  *       this list of conditions and the following disclaimer.          *
14  *                                                                      *
15  *     * Redistributions in binary form must reproduce the above copyright      *
16  *       notice, this list of conditions and the following disclaimer in the    *
17  *       documentation and/or other materials provided with the distribution.   *
18  *                                                                      *
19  *     * Neither the name of RPISEC nor the names of its contributors may be    *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                             *
22  *                                                                      *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED       *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF         *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN      *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,   *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED     *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR       *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF       *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING         *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS           *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.                 *
33  *                                                                      *
34  ************************************************************************/
35  #ifndef DEBUG_H
36  #define DEBUG_H
37
38  #define ERROR     0
39  #define WARINING  1
40  #define MESSAGE   2
41  #define LOG       3
42  #define DEBUG     4
43
44  #ifdef _DEBUG
45
46  #define _CODE(code) \
47  do { code } while(0)
48
49  #define _LOCKED_CODE(code) \
50  _CODE(MutexLock _log_locker(_log_mutex); code)
51
52  #define B_BEGIN(level, type) \
53  if((level) <= _log_level) cout << "<message_type=\"" << type << "\"_thread=\"" << (int) Thread::Self()<< "\">
54
55  #define B_ENTER(level, class, method) \
56  _LOCKED_CODE(if((level) <= _log_level) cout << "<message_type=\"ENTER\"_thread=\"" << (int) Thread::Self()<<
57
58
59  #define B_END(level) \
60  if((level) <= _log_level) cout << "</message_thread=\"" << (int) Thread::Self()<< "\">" << endl
61
62
63  #   define B_LOG(level, i, str)                          \
64  _LOCKED_CODE(                                            \
65    B_BEGIN(level, i);                                     \
66    if((level) <= _log_level) cout << str << endl; \
67    B_END(level);                                          \
68  )
69
70  #   define DO_ENTER(class, str)      _local_method_log(class, str)
71
72  #   define DO_ERROR(str)             B_LOG(ERROR, "ERROR", str)
73  #   define DO_WARNING(str)           B_LOG(WARNING, "WARNING", str)
74  #   define DO_MESSAGE(str)           B_LOG(MESSAGE, "MESSAGE", str)
75  #   define DO_LOG(str)               B_LOG(LOG, "LOG", str)
76  #   define DO_DEBUG(str)             B_LOG(DEBUG, "DEBUG", str)
77
78  #   define INIT_LOG(lvl)             Mutex _log_mutex; unsigned int _log_level = (lvl)
79
80  #include "Mutex.h"
81  #include "Thread.h"
82
83  extern Mutex _log_mutex;
84
85  #include <string>
86  #include <iostream>
87  using namespace std;
```

```
88
89  extern unsigned int _log_level;
90
91  class _local_method_log
92  {
93  public:
94    _local_method_log(string cl, string method) : m_cl(cl), m_method(method)
95    {
96      //B_LOG(DEBUG, "ENTER", m_cl + "::" + m_method);
97      B_ENTER(DEBUG, cl, method);
98    }
99
100   ~_local_method_log()
101   {
102     //B_LOG(DEBUG, "EXIT", m_cl + "::" + m_method);
103     _LOCKED_CODE(B_END(DEBUG););
104   }
105 private:
106   string m_cl;
107   string m_method;
108 };
109
110 #else
111 #   define B_LOG(level, i, str)
112 #   define DO_ENTER(class, str)
113 #   define DO_ERROR(str)
114 #   define DO_WARNING(str)
115 #   define DO_MESSAGE(str)
116 #   define DO_LOG(str)
117 #   define DO_DEBUG(str)
118 #   define INIT_LOG(lvl)
119
120 #endif /* _DEBUG */
121
122
123 #endif
```

E.2.12. v3/agent/DummyPlugin.h

```
1   /*************************************************************************
2   *                                                                        *
3   * Distributed Hash Cracker v3.0                                          *
4   *                                                                        *
5   * Copyright (c) 2009 RPISEC.                                             *
6   * Copyright (C) 2010 Samuel Rodriguez Sevilla                            *
7   * All rights reserved.                                                   *
8   *                                                                        *
9   * Redistribution and use in source and binary forms, with or without modifi-  *
10  * cation, are permitted provided that the following conditions are met:       *
11  *                                                                        *
12  *     * Redistributions of source code must retain the above copyright notice  *
13  *       this list of conditions and the following disclaimer.            *
14  *                                                                        *
15  *     * Redistributions in binary form must reproduce the above copyright      *
16  *       notice, this list of conditions and the following disclaimer in the    *
17  *       documentation and/or other materials provided with the distribution.   *
18  *                                                                        *
19  *     * Neither the name of RPISEC nor the names of its contributors may be    *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                               *
22  *                                                                        *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED       *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED    *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR      *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF      *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING        *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS          *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33  *                                                                        *
34  *************************************************************************/
35  #ifndef __DUMMY_PLUGIN_H__
36  #define __DUMMY_PLUGIN_H__
```

```
37
38 #include "Plugin.h"
39 #include "Algorithm.h"
40
41 class DummyAlgorithm : public Algorithm
42 {
43   /*!
44    *  @brief Returns the name of the algorithm
45    *  @return The algorithm name
46    */
47   string GetName();
48   /*!
49    *  @brief Returns the length of a hash generated by the algorithm
50    *  @return The generated hash length
51    */
52   int  HashLength();
53   /*!
54    *  @brief Each algorithm requires an input (normally a hash) with
55    *  a concrete length. This method returns the length of this input.
56    *  @return The algorithm input length
57    */
58   int  InputLength();
59   /*!
60    *  @brief Executes the CPU version of the algorithm. It is useful
61    *  if the system is not GPU accelerated.
62    */
63   void ExecuteCPU(WorkUnit& wu, int nCore);
64 #ifdef CUDA_ENABLED
65   /*!
66    *  @brief Executes the GPU version of the algorithm.
67    *  @param pDevice Information about the GPU device
68    */
69   void ExecuteGPU(WorkUnit& wu, Device* pDevice, CudaContext* pContext);
70 #endif
71   /*!
72    *  @brief Returns if the algorithm can use GPU capabilities
73    *  @return Tue if the Algorithm can use a GPU
74    */
75   bool IsGPUCapable();
76   /*!
77    *  @brief Returns if the algorithm cas use GPU capabilites
78    *  @return True if the algorithm can use a CPU
79    */
80   bool IsCPUCapable();
81   /*!
82    *  @brief Prepares information to accelerate hash attack or to perform better the functionality
83    *  @param wo The workunit with the information to analize
84    */
85   //virtual void Prepare(WorkUnit & wo) = 0;
86 #ifdef CUDA_ENABLED
87   /*!
88    *  @brief Prepared information to accelerate hash attack or to improve the performance. It is
89    *         indicated for CUDA devices.
90    *  @param pDevice
91    *  @param pContext
92    *  @param wu
93    */
94 //  virtual void Prepare(Device *pDevice, CudaContext *pContext, WorkUnit *wu) = 0;
95 #endif
96   /*!
97    *  @brief This method frees the internal memory used during the execution process.
98    */
99 //  virtual void End() = 0;
100   /*!
101    *  @brief Number of registers used by the GPU version of the algorithm.
102    *  This value can be obtained using the cubin compilation of the GPU unit. If it is
103    *  unknown it returns 0.
104    *  @return The number of registers used
105    */
106   int RegistersUsed();
107   /*!
108    *  @brief Amount of constant memory used by the GPU version of the algorithm.
109    *  This value can be obtained using the cubin compilation of the GPU unit. If it is
110    *  unknown it return 0.
111    *  @return The amount of constant memory used
112    */
113   int ConstantMemoryUsed();
```

```
114     /*!
115      *  @brief Amount of shared memory used by the GPU version of the algorithm.
116      *  This value can be obtained using the cubin compilation of the GPU unit. If it is
117      *  unknown it return 0.
118      *  @return The amount of shared memory used
119      */
120     int SharedMemoryUsed();
121 };
122
123 #endif
```

E.2.13. v3/agent/DummyPlugin.cpp

```
1   /*****************************************************************************
2   *                                                                            *
3   * Distributed Hash Cracker v3.0                                              *
4   *                                                                            *
5   * Copyright (c) 2009 RPISEC.                                                 *
6   * Copyright (C) 2010 Samuel Rodriguez Sevilla                                *
7   * All rights reserved.                                                       *
8   *                                                                            *
9   * Redistribution and use in source and binary forms, with or without modifi- *
10  * cation, are permitted provided that the following conditions are met:      *
11  *                                                                            *
12  *     * Redistributions of source code must retain the above copyright notice *
13  *       this list of conditions and the following disclaimer.                *
14  *                                                                            *
15  *     * Redistributions in binary form must reproduce the above copyright    *
16  *       notice, this list of conditions and the following disclaimer in the  *
17  *       documentation and/or other materials provided with the distribution. *
18  *                                                                            *
19  *     * Neither the name of RPISEC nor the names of its contributors may be  *
20  *       used to endorse or promote products derived from this software without *
21  *       specific prior written permission.                                   *
22  *                                                                            *
23  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
24  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
25  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
26  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
27  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
28  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
29  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
30  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
31  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
32  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
33  *                                                                            *
34  *****************************************************************************/
35  #include "DummyPlugin.h"
36
37  /*!
38   *  @brief Returns the name of the algorithm
39   *  @return The algorithm name
40   */
41  string DummyAlgorithm::GetName()
42  {
43    return string("DummyPlug");
44  }
45  /*!
46   *  @brief Returns the length of a hash generated by the algorithm
47   *  @return The generated hash length
48   */
49  int  DummyAlgorithm::HashLength()
50  {
51    return 0;
52  }
53  /*!
54   *  @brief Each algorithm requires an input (normally a hash) with
55   *  a concrete length. This method returns the length of this input.
56   *  @return The algorithm input length
57   */
58  int  DummyAlgorithm::InputLength()
59  {
60    return 0;
61  }
62  /*!
```

```
63    *  @brief Executes the CPU version of the algorithm. It is useful
64    *  if the system is not GPU accelerated.
65    */
66   void DummyAlgorithm::ExecuteCPU(WorkUnit& wu, int nCore)
67   {
68
69   }
70   #ifdef CUDA_ENABLED
71   /*!
72    *  @brief Executes the GPU version of the algorithm.
73    *  @param pDevice Information about the GPU device
74    */
75   void DummyAlgorithm::ExecuteGPU(WorkUnit& wu, Device* pDevice, CudaContext* pContext)
76   {
77
78   }
79   #endif
80   /*!
81    *  @brief Returns if the algorithm can use GPU capabilities
82    *  @return Tue if the Algorithm can use a GPU
83    */
84   bool DummyAlgorithm::IsGPUCapable()
85   {
86     return true;
87   }
88   /*!
89    *  @brief Returns if the algorithm cas use GPU capabilites
90    *  @return True if the algorithm can use a CPU
91    */
92   bool DummyAlgorithm::IsCPUCapable()
93   {
94     return true;
95   }
96   /*!
97    *  @brief Prepares information to accelerate hash attack or to perform better the functionality
98    *  @param wo The workunit with the information to analize
99    */
100  //virtual void DummyAlgorithm::Prepare(WorkUnit & wo) = 0;
101  #ifdef CUDA_ENABLED
102  /*!
103   *  @brief Prepared information to accelerate hash attack or to improve the performance. It is
104   *         indicated for CUDA devices.
105   *  @param pDevice
106   *  @param pContext
107   *  @param wu
108   */
109  //  virtual void DummyAlgorithm::Prepare(Device *pDevice, CudaContext *pContext, WorkUnit *wu) = 0;
110  #endif
111  /*!
112   *  @brief This method frees the internal memory used during the execution process.
113   */
114  //  virtual void DummyAlgorithm::End() = 0;
115  /*!
116   *  @brief Number of registers used by the GPU version of the algorithm.
117   *  This value can be obtained using the cubin compilation of the GPU unit. If it is
118   *  unknown it returns 0.
119   *  @return The number of registers used
120   */
121  int DummyAlgorithm::RegistersUsed()
122  {
123    return 0;
124  }
125  /*!
126   *  @brief Amount of constant memory used by the GPU version of the algorithm.
127   *  This value can be obtained using the cubin compilation of the GPU unit. If it is
128   *  unknown it return 0.
129   *  @return The amount of constant memory used
130   */
131  int DummyAlgorithm::ConstantMemoryUsed()
132  {
133    return 0;
134  }
135  /*!
136   *  @brief Amount of shared memory used by the GPU version of the algorithm.
137   *  This value can be obtained using the cubin compilation of the GPU unit. If it is
138   *  unknown it return 0.
139   *  @return The amount of shared memory used
```

```
140   */
141  int DummyAlgorithm::SharedMemoryUsed()
142  {
143    return 0;
144  }
145
146
147  BEGIN_PLUGIN("Samuel_Rodriguez_Sevilla")
148    ADD_ALGORITHM(DummyAlgorithm, "Dummy", 1)
149  END_PLUGIN()
```

E.3. Códigos importantes del controlador

E.3.1. v3/controller/app/controllers/crackers_controller.php

```
1   <?php
2   /*****************************************************************************
3   *                                                                            *
4   * Distributed Hash Cracker v3.0                                              *
5   *                                                                            *
6   * Copyright (c) 2009 RPISEC.                                                 *
7   * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
8   * All rights reserved.                                                       *
9   *                                                                            *
10  * Redistribution and use in source and binary forms, with or without modifi- *
11  * cation, are permitted provided that the following conditions are met:      *
12  *                                                                            *
13  *     * Redistributions of source code must retain the above copyright notice *
14  *       this list of conditions and the following disclaimer.                *
15  *                                                                            *
16  *     * Redistributions in binary form must reproduce the above copyright    *
17  *       notice, this list of conditions and the following disclaimer in the  *
18  *       documentation and/or other materials provided with the distribution. *
19  *                                                                            *
20  *     * Neither the name of RPISEC nor the names of its contributors may be  *
21  *       used to endorse or promote products derived from this software without *
22  *       specific prior written permission.                                   *
23  *                                                                            *
24  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
25  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
26  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
27  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
28  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
29  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
30  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
31  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
32  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
33  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
34  *                                                                            *
35  *****************************************************************************/
36
37  class CrackersController extends AppController {
38    var $name = 'Crackers';
39    var $uses = array('Stat', 'Crack', 'Hash', 'WorkUnit');
40    var $helpers = array('Html', 'Form', 'Session');
41
42    function index() {
43
44    }
45
46    function queue() {
47      $this->set('cracks', $this->Crack->findAllByActive(1));
48    }
49
50    function overview() {
51      //$this->layout = 'cracker';
52
53      $now = time();
54      $exp = $now - 300;
55
56      //dbquery("DELETE FROM `stats` WHERE `updated` < '$exp'");
57      $this->Stat->deleteAll(array('Stat.updated <' => '$exp'));
```

```
58
59      }
60
61      function submit() {
62        if(!empty($this->data)) {
63          $this->set('test', $this->data);
64
65          $alg = $this->data['Cracker']['alg'];
66          $hashes = explode("\n", $this->data['Cracker']['hashes']);
67          $len = intval($this->data['Cracker']['len']);
68          $exp = intval($this->data['Cracker']['exp']);
69          $pri = intval($this->data['Cracker']['pri']);
70
71          //Should never have any angle brackets but make sure
72          for($i=0; $i<count($hashes); $i++)
73            $hashes[$i] = strip_tags($hashes[$i]);
74              //Validate input
75          $bOK = true;
76          if($pri < 0 || $pri > 10)
77          {
78            $bOK = false;
79            $this->set('priError', __("Invalid priority level", true));
80          }
81          if($exp != 0)
82            $exp = time() + 60*$exp;
83          if($exp == 0)
84            $exp = time() + 10*365*24*60*60;
85          if($len < 0 || $len > 32)
86          {
87            $bOK = false;
88            $this->set('lenError', __("Invalid length", true));
89          }
90          $hash_lengths = array(
91            'md5' => 32,
92            'md4' => 32,
93            'ntlm' => 32,
94            'sha1' => 40,
95            'sha256' => 64,
96            'md5crypt' => -1  // TODO: do something here
97            );
98          $hashlen = $hash_lengths[$alg];
99          if($alg != 'md5crypt')
100         {
101           //Check length against array
102           //TODO: handle md5crypt variable salt lengths later
103           foreach($hashes as $hash)
104           {
105             $hash = trim($hash);
106             if(strlen($hash) != $hashlen)
107             {
108               $bOK = false;
109               $len = strlen($hash);
110               $error_msg = __("Invalid hash length %d on hash %s (expected %d)", true);
111               $this->set('hashesError', sprintf($error_msg, $len, $hash, $hashlen));
112             }
113           }
114         }
115         if(!isset($hash_lengths[$alg]))
116         {
117           $this->set('algError', __('Invalid hash algorithm', true));
118           $bOK = false;
119         }
120
121         //No batch cracking of md5crypt allowed since it's salted (we gain nothing)
122         if($alg == 'md5crypt' && count($hashes) != 1)
123         {
124           $this->set('algError', __("Batch cracking of md5crypt is not possible.", true));
125           $bOK = false;
126         }
127
128         //GPUs have limited memory!
129         if(count($hashes) > 128)
130         {
131           $this->set('hashesError', __("The current implementation is limited to 128 simultaneous hashes."));
132           $bOK = false;
133         }
134
```

```
135         if($bOK)
136         {
137           $alg = mysql_real_escape_string($alg);
138           $character_set = array(
139             'lower' => 'a',
140             'upper' => 'A',
141             'nums' => '1',
142             'somesymbols' => '!',
143             'allsymbols' => '>',
144             'space' => 's',
145             'newline' => 'n'
146           );
147
148           $cset = '';
149           foreach($this->data['Cracker']['chset'] as $ch) {
150             if(isset($character_set[$ch]))
151               $cset .= $character_set[$ch];
152           }
153
154           //Set next WU
155           $nwu = $cset[0];
156           if($nwu == 's')
157             $nwu = ' ';
158           if($nwu == 'n')
159             $nwu = '\n';
160
161           $ins_hashes = array();
162           foreach($hashes as $hash)
163           {
164             //Sanitize input
165             $hash = mysql_real_escape_string(trim($hash));
166
167             $ins_hashes[] = array(
168               'hash' => $hash,
169               'collision' => '',
170               'active' => '1');
171           }
172
173
174           $tm = time();
175           $this->Crack->saveAll(array(
176             'Crack' => array(
177               'algorithm' => $alg,
178               'charset' => $cset,
179               'maxlen' => $len,
180               'nextwu' => $nwu,
181               'started' => $tm,
182               'expiration' => $exp,
183               'priority' => $pri,
184               'active' => '1',
185               'updated' => $tm
186             ),
187             'Hash' => $ins_hashes
188           ));
189
190           $this->Session->setFlash(__('Information saved', true));
191         }
192
193       }
194     }
195
196     function output() {
197       $this->set('hashes', $this->Hash->findAllByActive(0, array(), array('Hash.id' => 'DESC')));
198     }
199
200     function stats() {
201       $this->set('stats', $this->Stat->find('all', array(
202         'order' => array(
203           'Stat.device' => 'ASC',
204           'Stat.updated' => 'DESC'
205         )
206       )));
207     }
208
209     function cancel($id) {
210       $this->Crack->delete($id);
211     }
```

```
212
213   function beforeFilter() {
214     parent::beforeFilter();
215
216     $this->set('action', $this->params['action']);
217     $this->layout = 'cracker';
218   }
219 }
220 ?>
```

E.3.2. v3/controller/app/controllers/agents_controller.php

```
1   <?php
2   /*****************************************************************************
3   *                                                                            *
4   * Distributed Hash Cracker v3.0                                              *
5   *                                                                            *
6   * Copyright (c) 2009 RPISEC.                                                 *
7   * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
8   * All rights reserved.                                                       *
9   *                                                                            *
10  * Redistribution and use in source and binary forms, with or without modifi- *
11  * cation, are permitted provided that the following conditions are met:      *
12  *                                                                            *
13  *     * Redistributions of source code must retain the above copyright notice *
14  *       this list of conditions and the following disclaimer.                *
15  *                                                                            *
16  *     * Redistributions in binary form must reproduce the above copyright    *
17  *       notice, this list of conditions and the following disclaimer in the  *
18  *       documentation and/or other materials provided with the distribution. *
19  *                                                                            *
20  *     * Neither the name of RPISEC nor the names of its contributors may be  *
21  *       used to endorse or promote products derived from this software without *
22  *       specific prior written permission.                                   *
23  *                                                                            *
24  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
25  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
26  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
27  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
28  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
29  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
30  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
31  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
32  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
33  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
34  *                                                                            *
35  *****************************************************************************/
36
37  class AgentsController extends AppController {
38    var $name = 'Agents';
39    var $uses = array('WorkUnit', 'Hash', 'Crack');
40    var $components = array('BaseN', 'Stats');
41    var $helpers = array('Xml');
42
43    function getwu() {
44      $this->layout = 'xml';
45
46      $hostname = mysql_real_escape_string($this->params['url']['hostname']);
47      $type = mysql_real_escape_string($this->params['url']['type']);
48      $num = intval($this->params['url']['num']);
49
50      $now = time();
51
52      $alglist = explode(',', $this->params['url']['accept-algorithms']);
53
54      // TODO: need to block Workunit, Crack and Hash?
55
56      // Look for all expired Workunits
57      $result = $this->WorkUnit->find('all',
58        array(
59          'conditions' => array('WorkUnit.expiration < ' => "$now"),
60          'order'      => array('WorkUnit.expiration ASC')
61        )
62      );
63
```

```
64      $orphaned_found = false;
65
66      foreach($result as $workunit) {
67        if(in_array($workunit['Crack']['algorithm'], $alglist)) {
68          // A expired WU with a compatible algorithm found!
69          $this->WorkUnit->updateAll(
70            array(
71              'WorkUnit.hostname' => "'$hostname'",
72              'WorkUnit.devtype' => "'$type'",
73              'WorkUnit.devid' => $num,
74              'WorkUnit.expiration' => time() + Configure::read('WorkUnit.expiration') /* two minutes */
75            ),
76            array('WorkUnit.id' => $workunit['WorkUnit']['id'])
77          );
78
79          $this->set('info', array(
80            'id' => $workunit['WorkUnit']['id'],
81            'algorithm' => $workunit['Crack']['algorithm'],
82            'charset' => $workunit['Crack']['charset'],
83            'start' => $workunit['WorkUnit']['start'],
84            'end' => $workunit['WorkUnit']['end'],
85            'hashes' => $this->Hash->find('list', array(
86              'fields' => array('Hash.id', 'Hash.hash'),
87              'conditions' => array('Hash.crack_id' => $workunit['Crack']['id'])
88            ))
89          ));
90
91          $orphaned_found = true;
92          break;
93        }
94      }
95
96      if(!$orphaned_found) {
97        $crack = $this->Crack->find('first', array(
98          'conditions' => array(
99            'Crack.active' => 1,
100           'Crack.algorithm' => $alglist
101         ),
102         'order' => array(
103           'Crack.priority DESC',
104           'Crack.started ASC'
105         )
106       ));
107 //$this->set('test', $crack);
108       if(count($crack['Crack']) < 1) {
109         $this->render('nowork'); // <nowork reason='idle'></nowork
110         return;
111       }
112
113
114
115       //Get our character set
116       $ccode = $crack['Crack']['charset'];
117       $csets = array(
118         'a' => "abcdefghijklmnopqrstuvwxyz",
119         'A' => "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
120         '1' => "1234567890",
121         '!' => "!@_-?#$",
122         '>' => "`~%^&*()=+[]\\{}|;':\",./<>",
123         's' => "_",
124         'n' => "\n"
125         );
126       $charset = '';
127       for($i = 0; $i < strlen($ccode); $i++)
128         $charset .= $csets[$ccode[$i]];
129       $base = strlen($charset);
130
131       //Make reverse charset
132       $rcharset = array();
133       for($i = 0; $i < $base; $i++)
134         $rcharset[$charset[$i]] = $i;
135
136       //Convert our starting position to a base-N integer
137       $nstart = $this->BaseN->make($crack['Crack']['nextwu'], $rcharset);
138
139       //Calculate work unit size
140       $wlength = 1000000000;      //Default for GPUs
```

```
141         if($type == 'core')
142           $wlength = 50000000;     //CPUs get less
143         if($crack['Crack']['algorithm'] == 'md5crypt')
144           $wlength /= 1000;       //md5crypt gets much smaller WUs
145
146         //Calculate our ending position (inclusive)
147         $nend = $this->BaseN->add($nstart, $base, $wlength);
148
149         //Calculate the start of the next work unit
150         //$cid = $crack->id;
151         $bSaturated = true;
152         $nwu = array();
153         for($i = 0; $i < count($nend); $i++)
154         {
155           if($nend[$i] != $base-1)
156             $bSaturated = false;
157         }
158         if($bSaturated)
159         {
160           //If we saturated, bump length and do {0...0}.
161           for($i = 0; $i < 1 + count($nend); $i++)
162             $nwu[$i] = 0;
163
164           //If we saturated and are at the end of the search space, mark the crack as exhausted.
165           //(If a work unit comes back with success in one of the last WUs, we can change it)
166           if(count($nend) == $crack['Crack']['maxlen'])
167           {
168             $crack['Crack']['active'] = 0;
169             $crack['Crack']['updated'] = time();
170             foreach($crack['Hash'] as $hash)
171               $hash['active'] = 0;
172           }
173         }
174         else
175         {
176           //If not, just add 1.
177           $nwu = $this->BaseN->add($nend, $base, 1);
178         }
179         $start = $crack['Crack']['nextwu'];
180         $end = $this->BaseN->toString($nend, $charset);
181
182         //Save start of next WU in table
183         $crack['Crack']['nextwu'] = mysql_real_escape_string($this->BaseN->toString($nwu, $charset));
184         $crack['Crack']['updated'] = time();
185
186         //Insert the new WU into the table
187         $now = time();
188         //$exp = $now + 120;  //TODO: is 2 minute expiration reasonable?
189         $workunit = array('WorkUit' => array(
190           'crack_id' => $crack['Crack']['id'],
191           'hostname' => $hostname,
192           'devtype' => $type,
193           'devid' => $num,
194           'start' => $start,
195           'end' => $end,
196           'started' => time(),
197           'expiration' => time() + Configure::read('WorkUnit.expiration') /*120*/ //TODO: is 2 minute expiratio
198         ));
199
200         $this->Crack->save($crack);
201         $this->WorkUnit->save($workunit);
202
203         $this->log( "\n" .
204           "id_________=_" . $this->WorkUnit->id . "\n" .
205           'algorithm_=_' . $crack['Crack']['algorithm'] . "\n" .
206           'charset____=_' . $crack['Crack']['charset'] . "\n" .
207           'start______=_' . $workunit['WorkUnit']['start'] . "\n" .
208           'end________=_' . $workunit['WorkUnit']['end'] . "\n" .
209           'hashes_____=_' . $crack['Hash'], 'debug'
210         );
211
212         $this->set('info', array(
213           'id' => $this->WorkUnit->id,
214           'algorithm' => $crack['Crack']['algorithm'],
215           'charset' => $crack['Crack']['charset'],
216           'start' => $workunit['WorkUnit']['start'],
217           'end' => $workunit['WorkUnit']['end'],
```

```
218           'hashes' => $crack['Hash']
219         ));
220       }
221     }
222
223
224
225
226
227
228     function submitwu() {
229       $this->layout = 'empty';
230
231       //dbquery("LOCK TABLES cracks WRITE, workunits WRITE, stats WRITE, history WRITE, hashes WRITE");
232       $dt = floatval($this->params['url']["dt"]);
233       $speed = floatval($this->params['url']["speed"]);
234
235       $id = intval($this->params['url']["wuid"]);
236       $collisions = intval($this->params['url']["collisions"]);
237
238       $hashes = array();
239       $cleartext = array();
240       for($i=0; $i<$collisions; $i++)
241       {
242         $hashes[$i] = intval($this->params['url']["hash$i"]);
243         $cleartext[$i] = mysql_real_escape_string($this->params['url']["cleartext$i"]);
244       }
245
246       $this->log("Requesting WorkUnit with id = $id", 'debug');
247       //Get the crack ID
248       $workunit = $this->WorkUnit->findById($id);
249
250       $this->log("WorkUnit = " . array2string($workunit), 'debug');
251       if(empty($workunit)) return;
252
253       $cid = $workunit['Crack']['id'];
254
255       $this->Stats->deleteOld();
256
257       //Get stats
258       $host = mysql_real_escape_string($workunit['WorkUnit']['hostname']);
259       $type = mysql_real_escape_string($workunit['WorkUnit']['devtype']);
260       $dev = $workunit['WorkUnit']['devid'];
261
262       $this->Stats->updateHistory();
263
264       $now = time();
265
266
267       $this->Stats->addStat(array(
268         'updated' => $now,
269         'device' => "$host-$type-$dev",
270         'time' => $dt,
271         'speed' => $speed
272       ));
273
274       //Update the crack
275       $update_crack = array();
276       if($collisions > 0)
277       {
278         for($i=0; $i<$collisions; $i++)
279         {
280           $hid = $hashes[$i];
281           $collision = $cleartext[$i];
282           $this->Hash->updateAll(
283             array(
284               'Hash.collision' => "'$collision'",
285               'Hash.active' => 0
286             ),
287             array(
288               'Hash.id' => $hid
289             )
290           );
291         }
292         $update_crack['Crack.updated'] = $now;
293       }
294
```

```
295       $active_hash = $this->Hash->find('all', array('conditions' => array('Hash.active' => 1, 'Hash.crack_id' =>
296       if(empty($active_hash)) {
297         $update_crack['Crack.active'] = 0;
298       }
299
300       if(!empty($update_crack))
301         $this->Crack->updateAll($update_crack, array('Crack.id' => $cid));
302
303       //Delete the work unit AFTER updating the crack in case of a server segfault or something
304       //Repeating a WU is better than missing one
305       $this->log("Trying to delete WorkUnit id = $id", 'debug');
306       $this->WorkUnit->delete($id);
307       $this->log('Victory xD', 'debug');
308
309       //dbquery("UNLOCK TABLES");
310       $this->layout = 'empty';
311     }
312   }
313   ?>
```

E.3.3. v3/controller/app/controllers/ajax_controller.php

```
1   <?php
2   /*****************************************************************************
3   *                                                                            *
4   * Distributed Hash Cracker v3.0                                              *
5   *                                                                            *
6   * Copyright (c) 2009 RPISEC.                                                 *
7   * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
8   * All rights reserved.                                                       *
9   *                                                                            *
10  * Redistribution and use in source and binary forms, with or without modifi- *
11  * cation, are permitted provided that the following conditions are met:      *
12  *                                                                            *
13  *     * Redistributions of source code must retain the above copyright notice *
14  *       this list of conditions and the following disclaimer.                *
15  *                                                                            *
16  *     * Redistributions in binary form must reproduce the above copyright    *
17  *       notice, this list of conditions and the following disclaimer in the  *
18  *       documentation and/or other materials provided with the distribution. *
19  *                                                                            *
20  *     * Neither the name of RPISEC nor the names of its contributors may be  *
21  *       used to endorse or promote products derived from this software without *
22  *       specific prior written permission.                                   *
23  *                                                                            *
24  * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
25  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
26  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
27  * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
28  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
29  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
30  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
31  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
32  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
33  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.               *
34  *                                                                            *
35  *****************************************************************************/
36
37  class AjaxController extends AppController {
38    var $components = array('Stats','RequestHandler' );
39    var $uses = array();
40    var $helpers = array('Javascript');
41  //  var $helpers = array('Cache');
42
43  //  var $cacheAction = array(
44  //    'history/' => 10
45  //  );
46    function history() {
47      $this->Stats->deleteOldHistory();
48      $this->set('history', $this->Stats->getHistory());
49    }
50  }
51  ?>
```

E.3.4. v3/controller/app/controllers/components/stats.php

```
1  <?php
2  /*****************************************************************************
3  *                                                                            *
4  * Distributed Hash Cracker v3.0                                              *
5  *                                                                            *
6  * Copyright (c) 2009 RPISEC.                                                 *
7  * Copyright (C) 2010 Samuel Rodríguez Sevilla                                *
8  * All rights reserved.                                                       *
9  *                                                                            *
10 * Redistribution and use in source and binary forms, with or without modifi- *
11 * cation, are permitted provided that the following conditions are met:      *
12 *                                                                            *
13 *     * Redistributions of source code must retain the above copyright notice *
14 *       this list of conditions and the following disclaimer.                *
15 *                                                                            *
16 *     * Redistributions in binary form must reproduce the above copyright    *
17 *       notice, this list of conditions and the following disclaimer in the  *
18 *       documentation and/or other materials provided with the distribution. *
19 *                                                                            *
20 *     * Neither the name of RPISEC nor the names of its contributors may be  *
21 *       used to endorse or promote products derived from this software without *
22 *       specific prior written permission.                                   *
23 *                                                                            *
24 * THIS SOFTWARE IS PROVIDED BY RPISEC "AS IS" AND ANY EXPRESS OR IMPLIED     *
25 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF       *
26 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN    *
27 * NO EVENT SHALL RPISEC BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, *
28 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED   *
29 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR     *
30 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF     *
31 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING       *
32 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS         *
33 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.              *
34 *                                                                            *
35 *****************************************************************************/
36
37 class StatsComponent extends Object {
38   /**
39    * When the Stats component is initialized it sets the internal
40    * Stat and History attributes to the model. It is important to
41    * use it on each method.
42    */
43   function startup(&$controller) {
44     $this->Stat = ClassRegistry::init('Stat');
45     $this->History = ClassRegistry::init('History');
46   }
47
48   /**
49    * Deletes the old stat values and the old history values.
50    */
51   function deleteOld() {
52     $now = time();
53     $exp = $now - Configure::read('WorkUnit.expiration'); //120;
54     $exp2 = $now - Configure::read('History.old'); //900;
55
56     $this->Stat->deleteAll(array('updated <' => $exp));
57     $this->History->deleteAll(array('time <' => $exp2));
58   }
59
60   /**
61    * Only deletes the old history values.
62    */
63   function deleteOldHistory() {
64     $exp = time() - Configure::read('History.old');
65     $this->History->deleteAll(array('time <' => $exp));
66   }
67
68   /**
69    * Check the stat values to calculate the history.
70    */
71   function updateHistory() {
72     //See if we already have history for this timestamp
73     //TODO: Make this more efficient
74     $now = time();
```

```
75
76      $r = $this->History->find('first', array(
77        'conditions' => array('time' => $now)
78      ));
79      if(empty($r))
80      {
81        $fspeed = 0;
82        $stats = $this->Stat->find('list',
83          array(
84            'fields' => array('Stat.id', 'Stat.speed')
85          )
86        );
87        foreach($stats as $id => $speed) {
88          $fspeed += $speed;
89        }
90
91        //Add to global stats
92        $this->History->save(
93          array(
94            'History' => array(
95              'time' => $now,
96              'speed' => $fspeed
97            )
98          )
99        );
100     }
101   }
102
103   /**
104    * Return the whole history
105    */
106   function getHistory() {
107     return $this->History->find('all', array('order' => 'History.time ASC'));
108   }
109
110   function addStat($stat) {
111     $this->Stat->deleteAll(array('device' => $stat['device']));
112     //Add us to stats
113     $this->Stat->save(array('Stat' => $stat));
114   }
115 }
116 ?>
```

E.3.5. v3/controller/app/controllers/components/base_n.php

```
1   <?php
2   function array2string($arr) {
3     $str = "";
4     if(is_array($arr)) {
5       $str .= "( ";
6       $first = true;
7       foreach($arr as $idx => $value) {
8         if($first) {
9           $first = false;
10        }
11        else {
12          $str .= ", ";
13        }
14      $str .= "[$idx] = " . array2string($value);
15      }
16      $str .= ") ";
17    }
18    else
19    {
20      $str .= $arr;
21    }
22    return $str;
23  }
24
25  class BaseNComponent extends Object {
26    /*!
27    * @brief Parses a base-N object out of a string and a reverse character set
28    *
29    * @param $str Input string
30    * @param $rcharset Reverse character set
```

```
31      *
32      * @return Base-N index array
33      */
34      function make($str, $rcharset)
35      {
36        $this->log("[BaseNComponent::make] rcharset = " . array2string($rcharset), 'debug');
37        $this->log("[BaseNComponent::make] str = " . $str, 'debug');
38
39        $ret = array();
40        for($i = 0; $i < strlen($str); $i++)
41          $ret[$i] = $rcharset[$str[$i]];
42
43        $this->log("[BaseNComponent::make] ret = " . array2string($ret), 'debug');
44        return $ret;
45      }
46
47
48      /*!
49      * @brief Adds an integer to a base-N object (using ripple carry)
50      *
51      * @param $num Input array
52      * @param $base Base of $num
53      * @param $add Value to add to $num
54
55      * @return Base-N index array
56      */
57      function add($num, $base, $add)
58      {
59        $len = count($num);
60
61        //Get number of digits required to add this number
62        $maxplace = ceil( log($add) / log($base) );
63        $this->log(
64          "[BaseNComponent::add]" .
65          "\tnum = " . array2string($num) .
66          "\tbase = $base\n" .
67          "\tadd = $add\n" .
68          "\tlen = $len\n" .
69          "\tmaxplace = $maxplace", 'debug');
70
71        //Too much? Can't possibly fit, saturate and quit
72        if($maxplace > $len)
73        {
74          for($i = 0; $i < $len; $i++)
75            $num[$i] = $base - 1;
76          $this->log("[BaseNComponent::add] Saturate: " . array2string($num), 'debug');
77          return $num;
78        }
79
80        //No, we will fit. Bump the rightmost digit
81        $num[$len-1] += $add;
82
83        //Handle carries
84        $bOverflow = false;
85        $pvalue = 1;
86        for($i = $len - 1; $i >= 0; $i--)
87        {
88          //Less than base? We're done
89          if($num[$i] < $base)
90            break;
91
92          //Greater than base? Handle carry
93          $low = $num[$i] % $base;
94          $high = floor($num[$i] / $base);
95          $num[$i] = $low;
96
97          //Carry off end? Overflow
98          if($high != 0 && $i==0)
99          {
100           $bOverflow = true;
101           break;
102         }
103
104         $num[$i-1] += $high;
105       }
106
107       //Saturate in case of overflow
```

```
108       if($bOverflow)
109       {
110         for($i=0; $i<$len; $i++)
111           $num[$i] = $base-1;
112       }
113
114       //Done
115       $this->log("[BaseNComponent::add] return: " . array2string($num), 'debug');
116       return $num;
117     }
118
119     /*!
120     * @brief Converts a base-N integer to a string
121     *
122     * @param $num Input array
123     * @param $charset Character set
124     *
125     * @return Output string
126     */
127     function toString($num, $charset)
128     {
129       $str = '';
130       for($i=0; $i<count($num); $i++)
131         $str .= $charset[$num[$i]];
132       $this->log("[BaseNComponent::toString] return: " . $str, 'debug');
133       return $str;
134     }
135
136     /*!
137     * @brief Converts a base-N integer to a float
138     *
139     * @param $num Input array
140     * @param $charset Character set
141     *
142     * @return Output value
143     */
144     function toFloat($num, $charset)
145     {
146       $ret = 0;
147       $len = count($num);
148       $base = strlen($charset);
149
150       for($i=0; $i<$len; $i++)
151       {
152         $pval = pow($base, $len-1-$i);
153         $ret += ($num[$i]+1) * $pval;
154       }
155
156       return $ret;
157     }
158
159   }
160   ?>
```

Apéndice F

Contenido del CD

En el CD que se adjunta vienen los siguiente elementos:

Document.pdf Es la memoria del proyecto fin de carrera tal y como se encuentra impresa.

cracker/ Es una copia completa del repositorio de versiones. Contiene el código del agente, del controlador y la memoria además de todos los cambios que se han realizado desde el inicio del proyecto.

Apéndice G

Aviso legal

A pesar de no haberse utilizado las correspondientes marcas de copyright y marcas comerciales a lo largo de este texto, todos los nombre utilizados pertenecen a sus respectivos propietarios según la ley 17/2001, de 7 de diciembre, de Marcas.

Apéndice H

Agradecimientos

La consecución de este proyecto no habría sido posible sin el apoyo y ayuda de mucha gente. Por este motivo desearía dedicar un agradecimiento a:

- José María Sierra, porque sin su paciencia y dedicación al proyecto éste nunca habría podido ser posible.
- A mis amigos David, Victor y Paula, que aunque sé que no os he podido hacer mucho caso siempre estabais ahí cuando os necesitaba.
- A mis padres, por su apoyo a los largo de tantos años.
- A mi novia Sandra, que a pesar de saber que soy informático me quiere igual.
- A mis compañeros de física, por todas las risas que no echamos y por toda la ayuda que me han ofrecido.

Sé que me dejo a mucha gente en el tintero y espero que me lo podáis perdonar. Gracias a todos.

Bibliografía

[AMD06] AMD. AMD Introduces World's First Dedicated Enterprise Stream Processor. http://www.amd.com/us/press-releases/Pages/Press_Release_114146.aspx, 2006.

[BK04] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In *Advances in Cryptology – EUROCRYPT '04, Lecture Notes in Computer Science*, pages 401–418. Springer-Verlag, 2004.

[Gro08] Khronos Group. Khronos launches heterogeneous computing initiative. http://www.khronos.org/news/press/releases/khronos_launches_heterogeneous_computing_initiative/, 2008.

[LP87] Dennis Luciano and Gordon Prichett. Cryptology: From caesar ciphers to public-key cryptosystems. *The College Mathematics Journal, Vol*, 18:2–17, 1987.

[Ltd08] Elcomsoft Co. Ltd. Creating a new reality: cracking one billion passwords per second. http://www.elcomsoft.com/PR/edpr_081002_en.pdf, 2008.

[MHP00] Cameron Mcdonald, Philip Hawkes, and Josef Pieprzyk. Differential Path for SHA-1 with complexity. 2000.

[NIS08] NIST. FIPS 180-3. Secure Hash Standard. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf, October 2008.

[NVI07] NVIDIA. NVIDIA® CUDA™ Unleashes Power of GPU Computing. http://www.nvidia.com/object/IO_39918.html, 2007.

[NVI10] NVIDIA. NVIDIA CUDA C Programming Guide. http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf, 2010.

[Sec] RSA Security. PKCS #6: Extended-Certificate Syntax Standard: 2.4.6 What are some techniques against hash functions? http://www.rsa.com/rsalabs/node.asp?id=2205.

[Tec04] Extreme Tech. Software turns gpu into audio coprocessor. http://www.extremetech.com/article2/0,1558,1642832,00.asp?kc=ETRSS02129TX1K0000532, 2004.

[vOW94] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *Proceedings of the 2nd ACM Conference on Computer and communications security*, CCS '94, pages 210–218, New York, NY, USA, 1994. ACM.

[Zon09] Andrew Zonenberg. Distributed Hash Cracker: A cross-Platform GPU-Accelerated Password Recovery System. http://www.cs.rpi.edu/~zonena/papers/cracker.pdf, April 2009.